Iterative split-and-merge algorithm for vector quantization codebook generation

Timo Kaukoranta
University of Turku
Turku Centre for Computer Science
Department of Computer Science
Lemminkäisenkatu 14A
Turku 20520
Finland
E-mail: tkaukora@cs.utu.fi

Pasi Fränti
University of Joensuu
Department of Computer Science
P.O. Box 111
Joensuu 80101
Finland

Olli Nevalainen
University of Turku
Turku Centre for Computer Science
Department of Computer Science
Lemminkäisenkatu 14A
Turku 20520
Finland

Abstract. We propose a new iterative algorithm for the generation of a codebook in vector quantization. The algorithm starts with an initial codebook that is improved by a combination of merge and split operations. By merging small neighboring clusters, additional resources (codevectors) are released. These extra codevectors can be reallocated by splitting large clusters. This process can be iterated until no further improvement is achieved in the distortion of the codebook. Experimental results show that the proposed method performs well in comparison to other tested methods, including the generalized Lloyd algorithm (GLA) and two hierarchical methods. *© 1998 Society of Photo-Optical Instrumentation Engineers.* [S0091-3286(98)01110-6]

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

Paper 27127 received Dec. 29, 1997; revised manuscript received Apr. 28, 1998; accepted for publication Apr. 28, 1998.

1 Introduction

We consider the codebook generation problem involved in the design of a *vector quantizer*.[1] The aim is to find M *codevectors (codebook)* for a given set of N *training vectors (training set)* by minimizing the average pairwise distance between the training vectors and their representative codevectors. The vectors are assumed to belong to a K-dimensional Euclidean space. The question of the proper choice for the training set is not addressed here; the motivation is simply to select the best possible codebook for a given training set.

There are several established methods for generating a codebook.[1] The most cited and widely used is the *generalized Lloyd algorithm*[2] (GLA). It starts with an initial solution, which can be chosen arbitrarily. The existing solution is then improved iteratively using two optimality criteria in turn until a local minimum is reached. The algorithm is relatively easy to implement and it gives reasonable results in most cases. Unfortunately the algorithm makes only local changes to the original codebook and it thus gets stuck at the first local minimum. The quality of the final codebook is therefore highly dependent on the initialization.

A different approach is to build the codebook hierarchically. An *iterative splitting algorithm*[3,4] starts with a codebook of size 1, where the only codevector is the centroid of the entire training set. The codebook is then iteratively enlarged by a splitting procedure until it reaches the desired size. This top-down process gives results similar to or even better than the GLA, with a faster algorithm.[4] A drawback of this method is that it is optimized only locally. The earlier splitting stages may harm the later choices of the algorithm, and there is therefore no guarantee that the final codebook will be globally optimal.

Another hierarchical algorithm, the *pairwise nearest neighbor*[5] (PNN), uses an opposite, bottom-up approach to codebook generation. It starts by initializing a codebook where each training vector is considered as its own codevector. Two codevectors are merged at each step of the algorithm, and the process is repeated until the desired size of the codebook is reached. The method outperforms both the GLA and the iterative splitting method, but it is very slow, especially if the training set is much larger than the codebook ($N \gg M$). There is also a faster variant, the *fast PNN*, but it weakens the results.[5]

Here we propose a new iterative algorithm for codebook generation, which combines the ideas of the two hierarchical methods and the GLA. The algorithm starts with an initial codebook, which is improved by a sequence of merge and split operations. Each iteration consists of two steps: The merge phase combines two nearby clusters (codevectors) similarly as in the PNN. The splitting phase divides a large cluster (codevector) into two subclusters using the split operation of Ref. 4. This two-step process is iterated until the codebook shows no further improvement.

The codevectors are considered as resources, released by the merge operation and reallocated by the split operation. The algorithm resembles the GLA inasmuch as it iteratively improves an existing codebook. The difference lies in the way the codebook is modified. The effect of the GLA is local, whereas the split-and-merge algorithm generates nonlocal changes to the codebook. By a local change we mean an operation that makes only minor modifications to the codevectors. The GLA, for example, does not change the general configuration of the codevectors during the iteration, and therefore the quality of the final codebook strongly depends on the initial codebook. The split-and-merge algorithm, on the other hand, performs global changes, which break the previous settlement of the codebook.

The proposed algorithm combines the benefits of both the iterative and the hierarchical approaches. We achieve the effectiveness of the PNN, but with a faster algorithm; the merge operations are not performed for large codebooks of size $O(N)$, but the algorithm starts directly from a codebook of the final size M. Therefore each merge operation can be performed in $O(M^2)$ time instead of the $O(N^2)$ time of the PNN. Because of the iterative approach, the results can be even better than for the PNN. Our experiments show that the new method outperforms the GLA, the PNN and the iterative splitting algorithm for all tested training sets.

The rest of the paper is organized as follows. The codebook generation problem is formally defined in Section 2. The split-and-merge algorithm is introduced in Section 3. The split-and-merge phases, and other details of the implementation, are discussed in the same section. Test results are presented in Section 4, and conclusions are drawn in Section 5.

2 Background

Let us consider a set $T=\{\mathbf{X}^{(i)}|1\leq i\leq N\}$ of training vectors in a K-dimensional Euclidean space. The aim is to find M codevectors $\mathbf{Y}^{(j)}$ $(1\leq j\leq M)$ by minimizing the average pairwise distance between the training vectors and their representative codevectors. The distance between two vectors $\mathbf{X}$ and $\mathbf{Y}$ is defined by their squared Euclidean distance:

$$\mathrm{d}(\mathbf{X},\mathbf{Y})=\sum_{k=1}^{K}(X_k-Y_k)^2, \tag{1}$$

where X_k and Y_k are the k'th components of the vectors. Let $C=\{\mathbf{Y}^{(j)}|1\leq j\leq M\}$ be a codebook, and $Q:T\rightarrow C$ be a mapping of the training vectors to their representative codevectors in C. The mapping thus gives us the partition $S=\{S^{(j)}|1\leq j\leq M\}$ of T. Here $S^{(j)}=\{\mathbf{X}\in T|Q(\mathbf{X})=\mathbf{j}\}$ is the set of training vectors (cluster) that are mapped to codevector $\mathbf{j}$. The distortion of the codebook C is then defined by:

$$\mathrm{distortion}(C)=\frac{1}{N}\sum_{i=1}^{N}\mathrm{d}\{\mathbf{X}^{(i)},Q[\mathbf{X}^{(i)}]\}. \tag{2}$$

A solution for the codebook construction problem can thus be defined by the pair (C,Q). These two depend on each other in such a manner that if one of them is given, the other one can be optimally constructed according to Eq. (2). This is formalized in the following two optimality criteria.

Partition optimality. For a given codebook C, partition S is optimal if each training vector $\mathbf{X}^{(i)}$ is mapped to its nearest codevector in C according to Eq. (1):

$$S^{(j)}=\{\mathbf{X}^{(i)}|\mathrm{d}[\mathbf{X}^{(i)},\mathbf{Y}^{(j)}]\leq\mathrm{d}[\mathbf{X}^{(i)},\mathbf{Y}^{(h)}]; \quad 1\leq i\leq N, 1\leq h\leq M\}. \tag{3}$$

Codebook optimality. For a given partition S, the codebook C is optimal if the codevectors $\mathbf{Y}^{(j)}$ $(1\leq j\leq M)$ are selected as the centroids of the clusters $S^{(j)}\in S$:

$$Y_k^{(j)}=\frac{\Sigma_{X\in S^{(j)}}X_k}{|S^{(j)}|}, \quad 1\leq k\leq K, \tag{4}$$

where $|S^{(j)}|$ is the cardinality of $S^{(j)}$. The codevector of a cluster is thus the mean (*centroid*) of the training vectors in the cluster. Codebook generation usually concentrates on finding a proper codebook C and the mapping function Q is assumed to be optimally defined according to Eq. (3).

The descriptions of the GLA, the iterative splitting method and the PNN are given in Figures 1–3. The GLA is a variant of the *K-means* clustering methods.[6] It is probably the most widely used method, due to its simple implementation; the method is a straightforward application of the two optimality criteria of Eqs. (3) and (4). If a particular cluster becomes empty, we replace the orphan codevector by a training vector that has the maximal distance to its nearest codevector. The time complexity of the GLA is $O(NMG)$, where G is the number of iterations.

The idea of iterative splitting is also very simple, but the implementation details are more complicated. The quality of the codebook is closely dependent on the way the split operation is designed. Several alternatives were discussed in a recent publication.[4] The time complexity of the fastest variant is $O(N\cdot\log N\cdot\log M)$, or $O(NM)$ if an additional refinement phase is added. The method is highly appropri-

```
Generate a codebook $C_0$ by any algorithm.
Repeat
    Generate optimal partition $S_{i+1}$ for a given codebook $C_i$.
    Generate optimal codebook $C_{i+1}$ for a given partition $S_{i+1}$.
    Fill empty clusters.
Until no improvement achieved.
```

Fig. 1 Structure of the GLA algorithm.

```
Set $m$=1 and calculate the centroid of the training set.
Repeat
    Select cluster $S^{(j)}$ to be split.
    Split the cluster $S^{(j)}$; $m \leftarrow m+1$.
    Update partition $S$ and codebook $C$.
Until $m=M$.
```

Fig. 2 Structure of the iterative splitting algorithm.

```
Set each training vector as a codevector (generating a codebook of size $m=N$).
Repeat
    Select two clusters $S^{(a)}$ and $S^{(b)}$ to be merged.
    Merge the selected clusters; $m \leftarrow m-1$.
    Update partition $S$ and codebook $C$.
Until $m=M$.
```

Fig. 3 Structure of the PNN algorithm.

```
Generate an initial codebook by any algorithm.
Repeat
        Select a cluster to be split.
        Split the selected cluster.
        Select two clusters to be merged
        Merge the selected clusters
Until no improvement achieved.
```

Fig. 4 Structure of the basic SM algorithm.

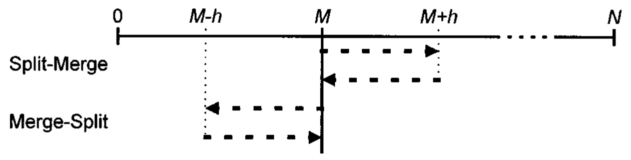

Fig. 5 Effect of the SM operations on the codebook size.

ate for the construction of a *tree structured vector quantizer*, since it produces a search tree for the codebook as a side-product.

The PNN is a variant of the *agglomerative clustering methods*.[7] A straightforward implementation uses local optimization for finding the clusters to be combined. The algorithm takes $O(N^3)$ time.[8] Fortunately, the time complexity can be reduced to $O(N^2)$ with rather simple modifications,[9] although the method is still significantly slower than the GLA and the iterative splitting method.

3 Iterative Split-and-Merge Algorithm

The general structure of the iterative split-and-merge (SM) algorithm is shown in Figure 4. In the merge phase, two nearby clusters are merged and thus one codevector is released from the codebook. In the split phase, a new codevector is inserted in the codebook. The order of these two phases can be freely chosen. Improvement is achieved if the split phase decreases the distortion more than the merge phase increases it. The size of the codebook remains the same before and after each iteration step. The method is iterated until no further improvement is achieved.

3.1 *Size and Order of the Iteration Step*

There are several alternative variants for the basic structure of the SM algorithm:

- ***One-split-one-merge:*** Perform a single split operation followed by a single merge operation (the algorithm in Figure 4).
- ***h-split-h-merge:*** Perform h split operations followed by h merge operations. The original size (M) of the codebook is preserved. Parameter h is called the *step size*.
- ***Adaptive split-and-merge:*** Perform a variable number of split and merge operations so that the size of the codebook converges finally to M.

The second approach makes more radical changes to the codebook during a single iteration step than the first approach. This gives a greater freedom in the design of the SM phases. For example, a faster algorithm is obtained by selecting $h=M$ and splitting all clusters at the same time. However, usually there are clusters whose split is either nonbeneficial or even impossible. It is therefore probably better to repeat the basic split operation M times. Another faster technique could be implemented by selecting the merged cluster pairs using a faster heuristic than the $O(N^2)$ PNN method.

The adaptive approach does not make any restrictions for the codebook size between the iterations, except that the final codebook must converge to the predefined size M. It is not obvious how the size of the codebook should be controlled. We therefore do not study this approach further in this paper.

Another question is the order of the SM phases, see Figure 5. Both of these operations are preceded by a selection step. For the merge operation, we select two clusters for which the merging is nonexpensive, and for the split step we chose a cluster that can beneficially be divided into two subclusters. It is therefore easy to see that in any practical situation, merge has no effect on the following split operation; if the cluster to be split were the same as the merged one, the iteration would have already come to a point where no further improvement can be achieved. On the other hand, it is possible that after split either of the new cluster halves may be merged with a third cluster. Therefore, if the algorithm starts by a series of splits it has more degrees of freedom in the merge operation. A minor drawback of this approach is that it is more time-consuming to perform split before merge than in the opposite order. Nevertheless, we treat the h-split-h-merge order as fixed. In the case of a one-split-one-merge approach, the preceding reasoning is of minor significance; in practice, both orders work similarly.

3.2 *Stopping Criterion*

A natural stopping criterion is to stop the iteration when no further improvement is observed. Let $D(t)$ be the distortion of Eq. (2) of the codebook after iteration round t. On one iteration round, the merge operation increases the total distortion by ΔD_M and the split operation decreases it by ΔD_S. Thus, the iteration improves the D-value if $\Delta D_S > \Delta D_M$.

Unlike in the GLA, there is no guarantee that the h-split-h-merge method will converge, but it may occur that $\Delta D_S < \Delta D_M$. It is also possible that even if the distortion of the current codebook increases, the following iterations can lead to a better solution. An alternative stopping criterion is therefore to use a fixed number of iterations and return the best codebook generated. Nevertheless, we use the greedy stopping criterion in the rest of the paper and stop at the first local minimum.

3.3 *Splitting Phase*

The aim of the splitting phase is to divide the training vectors of a large cluster into two subclusters and replace the original codevector by the centroids of the two subclusters. The operation involves two design questions: *the selection of the cluster to be split* and *the algorithm performing the division*. For the selection, we apply here a local optimization strategy where each cluster is tentatively split and the one decreasing the distortion most is chosen.

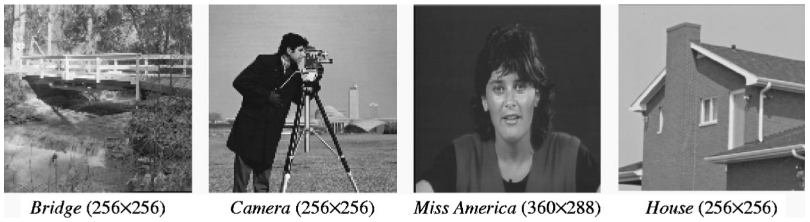

Fig. 6 Sources of the training sets.

The splitting is performed using an approximate variant of *principal component analysis* (PCA), as proposed in Ref. 4. The main idea of the PCA (Ref. 10, p. 8) is to calculate the principal axis of the training vectors in the cluster. The training vectors are classified by a $(K\text{-}1)$-dimensional hyperplane perpendicular to the principal axis, passing it at a point P (e.g., the cluster centroid). Instead of calculating the exact principal axis, we approximate its location for each dimension separately with respect to the dimension with the largest variance. This is done by solving first the dimension k with the largest variance, and then calculating the linear regression of the other dimensions with respect to the dimension k. The regression coefficients of these lines are considered as the direction of the approximate principal axis. The position P of the hyperplane is chosen optimally by considering each training vector as a tentative dividing point through which the hyper plane passes. The one with the lowest distortion value is selected.

The result of the split operation is fine-tuned by a partial remapping operation, as proposed in Ref. 4. This operation is based on the fact that some training vectors in the neighboring clusters may become closer to a new codevector than the original one. The partition boundaries can thus be improved by checking, for each training vector, whether one of the two new codevectors is closer than its current codevector. The codevectors are also recalculated after the remapping operation.

The time complexity of a single splitting phase is $O(N \cdot \log N)$, since there are M clusters to be tested and each tentative splitting requires $O[N/M \cdot \log(N/M)]$ time. Further splits $(h>1)$ can be performed in $O[N/M \cdot \log(N/M) \cdot \log h]$ time in total, because the calculations of the previous splitting step do not need to be repeated. The overall time complexity of the splitting phase is therefore $O(N \cdot \log N)+O(Nh)$. The latter term is due to the fine-tuning. For more details of the method and its design alternatives, see Ref. 4.

3.4 *Merge Phase*

The aim of the merge phase is to combine two nearby clusters and replace their corresponding codevectors by the centroid of the combined cluster. The merge operation thus releases one codevector from the codebook. We apply here the PNN method.[5] In this method the clusters to be merged are chosen by comparing all cluster pairs and taking the pair that least increases the total distortion. The increase in the distortion when merging clusters $S^{(i)}$ and $S^{(j)}$ is calculated as

$$\Delta D=\frac{n_i n_j}{n_i+n_j}\,\mathrm{d}[\mathbf{Y}^{(i)},\mathbf{Y}^{(j)}], \tag{5}$$

where n_i and n_j are the number of the training vectors in the clusters i and j. The centroid of the combined cluster is calculated as

$$\mathbf{Y}^{(\mathrm{new})}=\frac{n_i\mathbf{Y}^{(i)}+n_j\mathbf{Y}^{(j)}}{n_i+n_j}. \tag{6}$$

These calculations do not require any additional information besides what is already stored from each cluster. Time complexity of a single merge phase is $O(M^2)$ due to the selection. In a recent study,[9] however, it was shown that the calculations of preceding merging steps can be utilized in the next merge step, so that the time complexity of each following step is only $O(M)$. The overall time complexity of the merge phase is therefore $O(M^2)+O(Mh)$.

3.5 *Inclusion of the GLA*

The proposed SM algorithm and the GLA both improve an existing codebook iteratively. The algorithms have different approaches, however, SM performs global changes, whereas the GLA makes only local modifications to the codevectors. It is therefore sensible to combine the two approaches. It was proposed in Ref. 11 that a single step of a SM operation should be augmented as an integral part of the GLA. Here we take an opposite approach, and apply the GLA as a fine-tuning phase in the SM algorithm.

In principle, the SM algorithm first makes nonlocal changes to the codebook, which is then fine-tuned by the GLA. However, the greatest benefit is obtained from the GLA during the first few iterations. It is therefore sufficient to perform only two GLA iterations at the end of each iteration step. The inclusion of the GLA iterations has two positive effects: it improves the quality of the final codebook and it reduces the total number of iterations required. The latter benefit partially compensates for the increase in the running time.

4 Test Results

The following training sets are considered: ‘‘*Bridge*,’’ ‘‘*Camera*,’’ ‘‘*Miss America*,’’ and ‘‘*House*,’’ see Figure 6. The vectors in the first two sets (‘‘*Bridge*’’ and ‘‘*Camera*’’) are 4×4 pixel blocks taken from gray-scale images (8 bits/pixel). The third set (‘‘*Miss America*’’) was obtained by subtracting two subsequent image frames of the original

Table 1 Training sets and their statistics.

Training Set	Vector Dimensions (K)	Number of Training Vectors (N)	Number of Codevectors (M)
"*Bridge*"	16	4096	256
"*Camera*"	16	4096	256
"*Miss America*"	16	6480	256
"*House*"	3	65536	256

Table 2 Effect of the step size on the SM and SMG (for "*Bridge*").

	MSE		Number of Iterations		Time per Iteration (s)		Time in Total (s)	
Step Size (h)	SM	SMG	SM	SMG	SM	SMG	SM	SMG
1	176.49	168.69	122.9	46.9	0.5	4.6	64.7	217.5
2	175.74	168.42	67.1	26.8	0.6	4.8	40.9	127.3
4	175.73	168.50	40.2	17.5	0.7	4.9	30.1	85.7
8	174.77	168.64	28.1	11.7	1.0	5.2	27.9	61.0
16	173.38	167.72	21.8	11.1	1.4	5.6	31.0	61.9
32	171.33	166.79	17.7	12.1	2.2	6.3	39.6	76.4
64	168.99	165.93	16.9	12.4	3.8	7.8	63.4	96.5
128	167.25	164.77	14.9	11.5	6.9	10.9	103.2	125.2
256	166.18	163.81	10.8	8.8	13.5	17.3	145.5	152.2
512	166.23	163.58	8.6	6.5	28.0	31.8	240.7	206.8
1024	167.26	164.19	5.1	5.2	63.0	66.5	321.2	346.0
2048	168.51	165.25	3.7	3.0	158.2	162.6	585.2	487.7

video image sequence, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames have been used. Applications of this kind of data are found in video image compression.[12] The fourth data set ("*House*") consists of the RGB color tuples from the corresponding color image. This data could be applied for palette generation in color image quantization.[13] The data sets and their properties are summarized in Table 1. In the experiments made here, we will fix the number of clusters to $M=256$.

Two variants of the SM algorithm are considered: the basic algorithm with a fixed step size (SM), and the same algorithm augmented by two iterations of the GLA (SMG). In all tests, initial codebooks are generated by randomly picking M training vectors (without duplicates). Results are averages of 10 test runs, unless otherwise noted.

The effect of the step size h is studied in Table 2. The first split and the first merge operations are the most time consuming. Additional steps can be performed with much less computation. The increase of h causes a decrease in the number of iterations. The time-optimal step size is therefore not $h=1$, but it varies from 8 to 16, depending on the training set. The lowest mean square error (MSE) values are obtained with a step size varying from 128 to 512. The situation for "*Bridge*" is illustrated in Figure 7. Note that the upper limit for the step size is $h=N-M$. In this case the method reduces back to the standard PNN algorithm. In the following we fix the step size to $h=256$.

Figure 8 illustrates the development of the distortion as a function of the number of iterations. The arrows show the

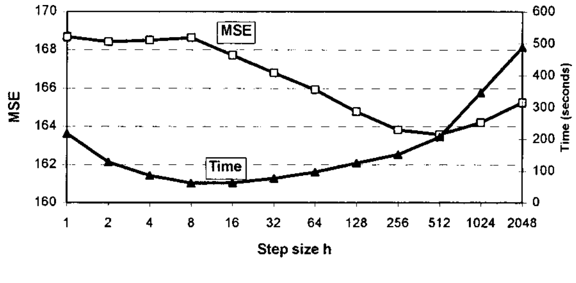

Fig. 7 MSE and running time of SMG as a function of the step size (for "*Bridge*").

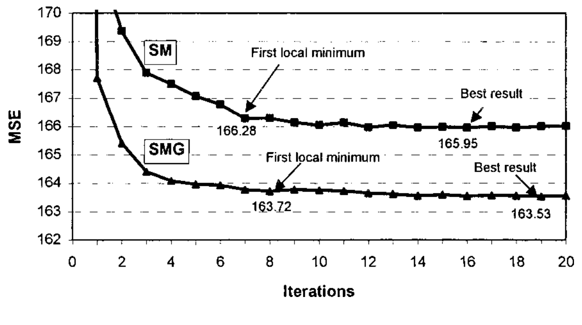

Fig. 8 Convergence of the SM and SMG algorithm (for "*Bridge*"). Step size is $h=256$.

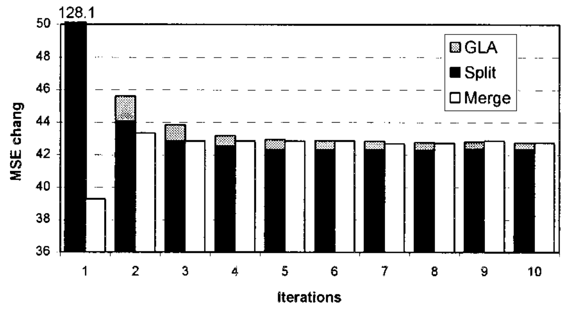

Fig. 9 Effect of the split, merge and GLA phases on MSE during the iteration (for "*Bridge*"). Step size is $h=256$.

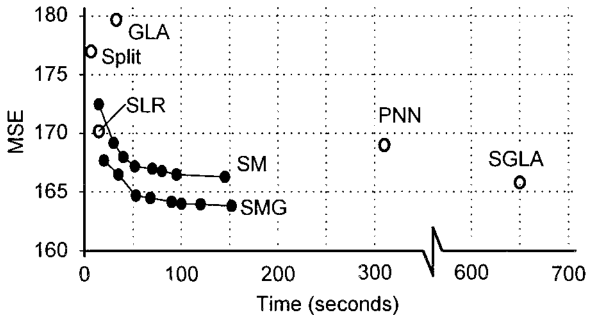

Fig. 10 Speed versus quality trade-off comparison of the main variants ("*Bridge*"). The SM and SMG lines represent the results after one to eight iterations.

Table 3 Comparison of MSE values for the various codebook generation methods.

Training Set	GLA	Split	SLR	SGLA	PNN	SM	SMG
"*Bridge*"	179.68	176.96	170.22	165.86	169.11	166.18	163.81
"*Camera*"	122.61	76.55	72.98	71.05	71.04	70.16	69.40
"*Miss America*"	5.96	5.54	5.40	5.28	5.51	5.26	5.19
"*House*"	7.81	6.46	6.18	6.05	6.37	6.17	5.98

Table 4 Comparison of the running times for the various codebook generation methods.

Training Set	GLA	Split	SLR	SGLA	PNN	SM	SMG
"*Bridge*"	33	7	16	651	311	146	152
"*Camera*"	38	6	14	473	284	104	96
"*Miss America*"	67	11	36	1089	829	231	285
"*House*"	254	25	70	2001	5649	173	358

points where the algorithm terminates if a greedy stopping criterion is applied. Slightly better results would be obtained if the iteration were continued further. Figure 9 illustrates the effects of the split, merge and GLA phases. The iteration continues as long as the overall effect of the split and GLA phases exceeds the effect of the merge phase.

We next compare the two SM algorithms (SM and SMG) with the following algorithms:

- *GLA:* the standard GLA method[2] using random initialization
- *Split:* iterative splitting algorithm of Ref. 3
- *SLR:* iterative splitting with an additional refinement phase[4]
- *SGLA:* the best splitting method of Ref. 4, where two GLA iterations are applied after each splitting phase
- *PNN:* the optimal PNN algorithm of Ref. 5.

The main results are summarized in Table 3 and Table 4. The SM and SMG give better MSE values than any of the preceding methods. A drawback of the SM and SMG, however, is the running time, which is about three times as long as required for the GLA. A faster version can be obtained if the total number of iterations is reduced (see Figure 10). The faster version gives similar or better time-distortion result than the GLA. If time is a critical factor, one should consider the use of the iterative splitting method (*Split*, *SLR*).

The effect of random initialization is illustrated in Figure 11. The proposed methods produce a narrow histogram of the MSE values (the standard deviation is 0.40 for SM and 0.32 for SMG), whereas the results for the GLA are more diverse (the standard deviation is 1.42). The SM algorithm is therefore less dependent on the initialization than the GLA.

5 Conclusions

A new algorithm was proposed for the codebook generation in vector quantization. The method starts with an initial codebook, which is iteratively improved by a sequence of merge and split operations. The proposed algorithm combines the benefits of both the iterative and the hierarchical approaches. The effectiveness of the PNN was achieved but with a faster algorithm. Because of the iterative approach, the results are even better than for the PNN.

The proposed SM algorithm outperforms all comparative methods. The results are about 10 to 40% better than that of the GLA. The method is also less sensitive to the random initialization, whereas the results of the GLA have much higher variation. The running time of the method is competitive with the GLA and much faster than that of the

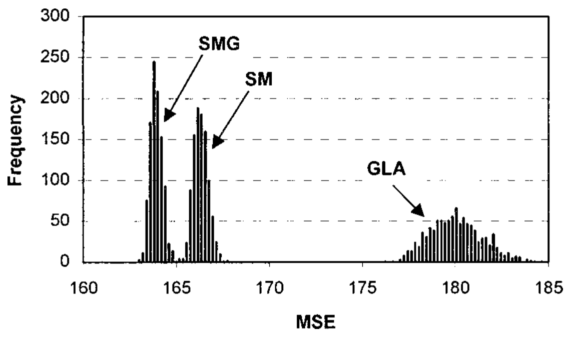

Fig. 11 Histograms of the final MSE values of the SM, SMG and GLA (for "*Bridge*"). Results are for 1000 runs.

PNN. In comparison to the splitting methods, SM and SMG give much better results, but at the cost of higher running times.

The SM approach includes a number of design alternatives. Because we were aiming at simplicity, we relegated some of these details to a brief discussion. The method of iteration used here (h-split-h-merge) is not the only possibility. The idea of oscillating between larger and smaller codebooks is worth further investigation. The same holds for stopping criterion. In our algorithms the SM operations are deterministic: the target clusters are selected in a greedy way. By adding more randomness to these operations, local minima could be evaded more effectively.

Acknowledgments

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

References

1. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht (1992).
2. Y. Linde, A. Buzo and R. M. Gray, ''An algorithm for vector quantizer design,'' *IEEE Trans. Commun.* **28**(1), 84–95 (1980).
3. X. Wu and K. Zhang, ''A better tree-structured vector quantizer,'' in *Proc. IEEE Data Compression Conference*, pp. 392–401, Snowbird, UT (1991).
4. P. Fränti, T. Kaukoranta and O. Nevalainen, ''On the splitting method for VQ codebook generation,'' *Opt. Eng.* **36**(11), 3043–3051 (1997).
5. W. H. Equitz, ''A new vector quantization clustering algorithm,'' *IEEE Trans. Acoust. Speech Signal Process.* **37**(10), 1568–1575 (1989).
6. J. B. McQueen, ''Some methods of classification and analysis of multivariate observations,'' in *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability,* Vol. 1, pp. 281–296, Univ. of California (1967).
7. K. M. Cunningham and J. C. Ogilvie, ''Evaluation of hierarchical grouping techniques, a preliminary study,'' *Comput. J.* **15**(3), 209–213 (1972).
8. J. Shanbehzadeh and P. O. Ogunbona, ''On the computational complexity of the LBG and PNN algorithms,'' *IEEE Trans. Image Process.* **6**(4), 614–616 (1997).
9. P. Fränti and T. Kaukoranta, ''Fast implementation of the optimal PNN method,'' in *Proc. IEEE Int. Conf. on Image Processing*, Chicago, IL (in press).
10. S. Kotz, N. L. Johnson and C. B. Read, Eds., *Encyclopedia of Statistical Sciences*, Vol. 6, John Wiley Sons, New York (1985).
11. T. Kaukoranta, P. Fränti and O. Nevalainen, ''Reallocation of GLA codevectors for evading local minima,'' *Electron. Lett.* **32**(17), 1563–1564 (1996).
12. J. E. Fowler Jr., M. R. Carbonara and S. C. Ahalt, ''Image coding using differential vector quantization,'' *IEEE Trans. Circ. Syst. Video Technol.* **3**(5), 350–367 (1993).
13. M. T. Orchard and C. A. Bouman, ''Color quantization of images,'' *IEEE Trans. Signal Process.* **39**(12), 2677–2690 (1991).

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

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

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, where since 1976 he has been an associate professor. He lectures in the areas of data structures, algorithm design and analysis, compiler construction and operating systems and his research interests are in the field of algorithm design, including image compression, scheduling algorithms and production planning.