Genetic Algorithms for Codebook Generation in Vector Quantization

Pasi Fränti*, Juha Kivijärvi#, Timo Kaukoranta# and Olli Nevalainen#

*Department of Computer Science,
University of Joensuu, PB 111
FIN-80101 Joensuu, FINLAND
Email: franti@cs.joensuu.fi

#Department of Computer Science
University of Turku
Lemminkäisenkatu 14 A
FIN-20520 Turku, FINLAND

Abstract

In the present paper we study genetic algorithms for the codebook generation problem of vector quantization. There are two different approaches to the problem: a codebook-based and a partition-based. Both these approaches can be derived from the optimality criteria of GLA. From these, the codebook-based approach is clearly superior to the partition-based approach. The experiments show that a pure genetic algorithm does not give comparable results to the existing methods, but the inclusion of GLA iterations is vital. Even in this case, the time-distortion performance of the algorithm is still inferior to simulated annealing. In the case of binary images, the hybrid combination of GA and GLA gives the best results. The greatest deficiency of the algorithm is still its high running time.

Indexing terms: vector quantization, codebook generation, image compression

1. Introduction

In the present paper we study the *codebook generation* of *vector quantization*. The aim is to find a *codebook* that would minimize the average distortion between a given *training set* and the codebook. It is assumed that the vectors are mapped to their nearest representative in the codebook in respect to a *distortion function*. The problem of generating optimal codebook, in its combinatorial form, has been shown to be NP-complete [1]. In other words, there is no known polynomial time algorithm to find the globally optimal solution, but reasonable suboptimal solutions are typically obtained by heuristic algorithms.

Non-parametric optimization techniques are applicable to the problem. A common property of these algorithms is that they iteratively try several possible solutions (codebooks) to the problem and at each step a new solution is generated by making modifications to the current one. For example, the well known *generalized Lloyd algorithm* (GLA) [2] starts with an initial codebook (which can be chosen randomly). The codebook is then improved iteratively by *hill-climbing strategy* until a local optimum is reached.

Genetic algorithms (GA) [3] are optimization techniques derived from the model of the natural selection in real life. The idea of genetic algorithms is that a *population* of possible solutions called *individuals* is first (randomly) generated. As in real life, only the best individuals survive. The next generation consists of the survivors and of new individuals generated from the survivors by applying genetic operations such as *crossover* and *mutation*.

For the codebook generation problem in VQ, two genetic algorithms have been proposed in literature by Delport and Koschorreck [4], and by Pan, McInnes and Jack [5]. These algorithms, however, are not pure genetic algorithms but hybrid combinations of GA and GLA. Moreover, the chosen parameters of GA are not analyzed in these studies; thus the question of the real benefits of genetic algorithms should still be considered.

In the present paper we introduce a general framework of GA into the codebook generation problem. The algorithm consist of three main phases: *selection*, *crossover*, and *refinement by GLA*. This helps us to understand the situation better than if studied only as a highly integrated entity. For example, the crossover operation in GA can be considered as a special case of the codebook generation problem itself. This knowledge immediately implies several possible solutions to the crossover problem. Several new algorithms will indeed be introduced here.

The application of vector quantization is here *image compression in spatial domain*, whereas in the previous studies it was applied to vectors of DCT-coefficients [4], or speech coding [5]. In the compression phase the image is divided into 4×4-non-overlapping blocks which are coded separately. For each block its nearest representative is searched from the codebook by any (optimal or suboptimal) *mapping function*. The index of the chosen codevector is then sent to the decoder, which replaces the original block by its representative in the codebook.

There are two different approaches to the codebook generation problem: a *codebook-based* [5], and a *partition-based* [4] approach. Both these approaches can be derived from the optimality criteria of GLA. From these, the codebook-based approach is clearly superior to the partition-based. Moreover, the experiments will show that a pure genetic algorithm is competitive with the existing methods only in the case of binary images. For gray scale images, the inclusion of GLA iterations is vital.

The rest of the paper is organized as follows. We start in Section 2 by defining the basic concepts of the problem and by giving a brief review of the previous research on the codebook generation methods by non-parametric optimization techniques. A sketch of the genetic algorithm is then given in Section 3. Each phase of the general algorithm is studied separately and results of the test runs will be given for the different variants of the algorithm. The best combinations of the genetic algorithm are then compared with the existing algorithms in Section 4. The algorithm is tested with training sets obtained from binary and gray scale images. Finally, conclusions are drawn in Section 5.

2. Background

Let us consider training vectors in a K-dimensional Euclidean space. Denote the number of vectors in the training set by N. Vector quantization partitions the whole input space into M non-overlapping regions. A representative (*codevector*) is assigned to each partition (*cluster*). The aim is to minimize the average distance (*distortion*) between the training vectors and their representatives. The question of the proper choice for the training set is not issued here. The motivation is merely to select the best possible codebook for the given training set no matter how it has been chosen. The distance between two vectors in an Euclidean space is measured by L_2-metrics:

$$d(X,Y) = \sqrt{\sum_{i=1}^{K} \left(X_i - Y_i\right)^2} \qquad \textbf{(1)}$$

where X and Y are two K-dimensional vectors, and X_i refers to the i'th component of the vector. Other distortion measures could be applied instead of the Euclidean distance as well. For example, in the case of image compression one could consider more complex measures that calculate the difference in the visual quality in the image blocks [6].

The average distortion caused by the VQ is calculated by the *mean square error* (MSE):

$$MSE = \frac{1}{NK} \sum_{i=1}^{N} d\left(X_i, f\left(X_i\right)\right)^2 \qquad \textbf{(2)}$$

where $f(X)$ is a mapping function from training vector X to its representative codevector in the codebook. The mapping function $f(X)$ defines the partitioning of the training set. In the MSE calculations we assume that vectors are mapped to their nearest representative (*optimal mapping*) in the codebook, i.e. the one minimizing the given distortion function. When the final codebook has been created, it is possible to replace the optimal mapping function by an approximative function for gaining speed. For example, *tree-structured codebook organization* allows $O(\log(M))$ time mapping (*heuristic search*) compared to the $O(M)$ time of the optimal mapping (*full search*) [7].

The codebook generation is considered here as a search problem. Thus, *non-parametric optimization techniques* can be applied to the problem. In abstract level these techniques are applicable to a great variety of search problems including the clustering problem in general. A common property of these algorithms is that they iteratively try several possible solutions to the problem and at each step a new *neighboring* solution, or a set of solutions is generated by making (small) modifications to the current solution. The initial solution can be generated by any existing algorithm and even a random solution is usually acceptable. The key questions in the optimization are

- The search strategy
- The definition of a neighboring solution

The solution is sometimes represented as a bit string and the neighboring solutions as any solution that differs from the original one in a fixed number of bit positions. We prefer here a problem oriented representation where the solution is defined by the pair (*partitioning*, *codebook*). Partitioning describes mapping from the training set to the codebook, and the codebook describes the codevectors. Now the concept of neighboring solution must be defined by a problem specific algorithm. Let us briefly discuss the following search strategies and their application to the codebook generation problem:

- Hill-climbing strategy
- Simulated annealing
- Tabu search
- Genetic algorithm

Hill-climbing strategy:

GLA [2] is a localized search technique that uses basically a very simple hill-climbing strategy (or perhaps *valley-descending strategy* due to the minimization problem) based on the two optimality criteria:

- *Nearest neighbor condition*: For a given codebook, the optimal partitioning of the training set can be obtained by mapping each training vector to its nearest codevector in the codebook, in respect to the distortion function.

- *Centroid condition*: For a given partition, its optimal codevector is the *centroid* (average vector) of the vectors within the partition.

The nearest neighbor condition clearly defines the optimal partitioning. It can be easily constructed by mapping each training vector to the codevector that minimizes the distortion function. This is a computationally expensive operation taking $O(NM)$ time. The centroid calculation, on the other hand, takes only $O(N)$ time.

In GLA, the candidate solution is iteratively modified by applying the two optimality criteria in turn. In the first stage the codebook is fixed and the partitions are recalculated using the nearest neighbor rule according to the codebook of the current solution. In the second stage the partitions are fixed and a new codebook is obtained by calculating the centroids of the partitions. The optimality criteria guarantees that the new solution is always at least as good as the previous one. The process is iterated until no improvement is achieved, or the number of iterations reaches a predefined limit.

Simulated annealing:

GLA uses a search technique based on a greedy heuristic where the minimization function is never allowed to increase. Thus the algorithm will get stuck to a local optimum which may be far from the global optimum. A variant of GLA, referred as *simulated annealing* (SA), tries to remove this drawback by a stochastic relaxation technique. At each step random noise is added to codevectors [8], or to training

vectors [9] so that the algorithm can jump over the local optimum. The amount of noise gradually decreases at each iteration step and eventually, when the amount of noise is completely eliminated the algorithm converges to another local optimum, which hopefully is a better solution to the problem. The time complexity of one SA iteration is equal to that of GLA but the total number of iterations is greater, and the inclusion of noise needs extra computational efforts in each phase.

Tabu search:

Tabu search is different from the hill-climbing technique in the sense that new solution in each iteration round does not need to be better than the previous one. Another difference is that several candidate solutions are considered in each phase, whereas GLA and SA algorithms proceed by using the nearest neighbor and centroid conditions as the optimality criteria. From these candidates, the best one is chosen, even if it is worse than the current one. The tabu search algorithm uses so-called *tabu list* to prevent the search from coming back to solutions that have already been visited. The tabu list includes certain number of previously visited solutions.

Let us next briefly discuss the algorithm proposed in [10] for the *clustering problem*. The algorithm manipulates the partitions of training vectors. The initial solution is chosen randomly. New candidate solutions are generated by modifying the partitions in a random fashion; a training vector is moved from the original partition to another (randomly chosen) partition with a probability of p (0.05 in [10]). The centroid condition was used to select the codevectors for each partition.

It is not clear whether the above algorithm would apply directly to the codebook generation problem in vector quantization. The search space in vector quantization is significantly larger than in the application considered in [10]. The M partitions can be chosen in M^N different ways. With the parameters (N=4096, M=256) the size of the search space is thus 2^{32768} ($\approx 10^{9864}$) compared to the size of 2^{40} ($\approx 10^{12}$) of the standard test problem used in [10] with parameters (N=40, M=2). Because of the huge search space it is unlikely that the use of a tabu list would have a major effect on the algorithm. Otherwise the algorithm resembles GA in the sense that they are both based on systematic randomizing. Hence, we will focus our study on the genetic algorithms in the rest of this paper.

3. Genetic algorithm

Genetic algorithms are iterative like the other non-parametric optimization techniques. Instead of generating only one solution in each iteration round GA generates several solutions. The *population* of solutions is generated from the best solutions of the previous *generation* by applying genetic operations such as *crossover* and *mutation*. Compare this to the tabu search where the candidates of the next round are generated on the basis of a single solution only, whereas genetic algorithms try to combine the properties of several (preferable good) candidate solutions.

In GA the individuals of the population are the candidate solutions. The solution of the codebook generation problem can be defined by the pair (*partitioning*, *codebook*).

For convenience, let us next assume that the solution is described by the codebook alone, and the partitioning by the optimal mapping given by the nearest neighbor condition. Here we adopt the *single chromosome representation* of the individuals, where a codevector appears as the *elementary unit* (*gene*). A codevector can never be splitted but it is always treated as an entity. The structure of the GA is given in Figure 1.

- Select the initial generation (G_0) of S codebooks.
- Iterate the following T times
  - Select the S_B surviving codebooks from G_i. Denote this set by G_{best}
  - Cross all possible pairs of surviving codebooks. Denote the resulting set of $S_B \cdot (S_B - 1) / 2$ codebooks by G_{new}
  - Generate mutations to the codevectors in G_{new}.
  - The next generation is $G_{i+1} = G_{best} \cup G_{new}$
- The solution of the algorithm is the best codebook in the final generation G_T.

Figure 1. Sketch of the genetic algorithm.

The codebooks of the initial population are created by selecting M random codevectors from the training set. Duplicate codevectors are not allowed. The initial population can also be chosen by any other codebook generation algorithm but we prefer a random starting point for now, see Section 4 for other possibilities. The algorithm is iterated T times.

The selection is performed by an *elitist* algorithm so that the S_B best codebooks always survive as such. The rest are discarded. The best individuals are defined by the so-called *fitness function* which is the *mean square error* (MSE) of the codebook. The elitism guarantees that the best solutions survive to the final population but it also allows fast convergence of the algorithm. This is important because of the high computational load of the partitioning phase, see Section 3.1.

The number of survivors is the primary parameter of the selection. If there are S_B survivors, then there are $S_B \cdot (S_B-1)/2$ new codebooks due to the crossover; and the size of the population is $S = S_B \cdot (S_B+1)/2$. For example, for $S_B=9$ the population size is $S=45$. There are several possibilities for performing the crossover but a random algorithm is once again a good alternative, especially with the elitist algorithm. Here the new codebook is generated by choosing $M/2$ random codevectors from each of the two parent codebooks (duplicates are excluded). For a discussion of various crossover algorithms, see Section 3.2.

The codevectors are mutated with a probability of P_m. In the mutation the codevector is replaced by a randomly chosen vector from the training set. The use of mutations is left as an option in the algorithm.

The main phases of GA are discussed in the next sections in greater detail. The computer simulations are performed using codebooks of M=256 vectors. The population size S=45 is a trade-off between time and distortion. The number of iterations is set to 50 in most of the experiments. If mutations are applied their probability is P_m=0.01.

3.1 Main variants of the algorithm

The two optimality criteria of GLA (nearest neighbor condition and centroid condition) depend on each other in the sense that if one of them is given, the optimal solution to the other one can be easily derived. This fact implies two alternative approaches to GA:

- codebook-based (CB)
- partition-based (PB)

In the *codebook-based* variant codebooks are the individuals of the population, and thus the objects of the genetic operations. Each codebook is represented by an M-length array of K-dimensional vectors, see Figure 2. The elementary unit (gene) is a codevector. Partitions (the mapping function) are calculated using the nearest neighbor condition. This is the most intuitive way to describe the problem and it is our main variant, see Figure 1.

In the *partition-based* variant the partitions are the individuals in the population. Each partition is represented as an N-length array of integers in the range $[1..M]$ defining mapping from the training set to the codebook, see Figure 2. The actual codebook is then calculated on the basis of the mapping using the centroid condition. The elementary unit (gene) is a single mapping value. The individual thus has N genes (compared to the CB-variant having M genes).

The CB-variant has the limitation that the codevectors are, not only limited to the training vectors, but also limited to the set of codevectors in the initial population. This restriction is due to the crossover operation that cannot create new codevectors but makes different combinations of the existing ones only. Mutation operation can alter the set of codevectors in the population but it is also limited to the training vectors. PB-variant does not suffer from these restrictions. The results of the CB-variant, however, are superior to those of the PB-variant as the results will later show. Moreover, we shall see in Section 4 that these restrictions disappear when the genetic algorithm is augmented with GLA.

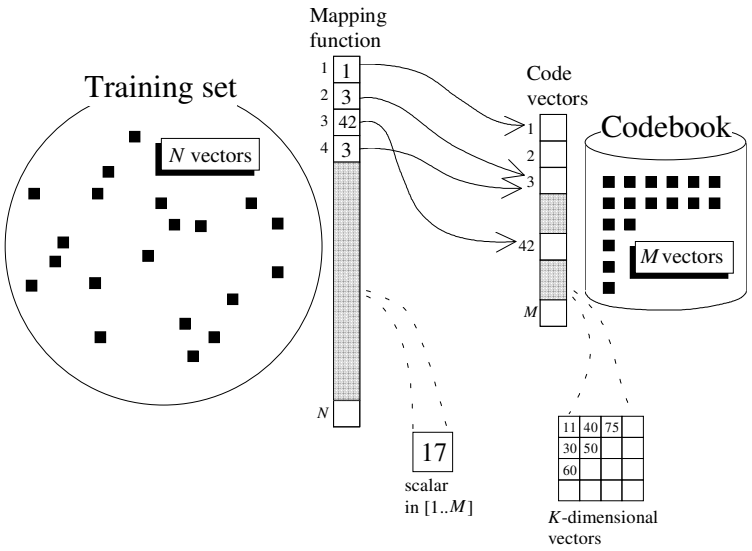

Figure 2. Representation of a training set and a codebook.

So far we have assumed that the selection phase is performed by the elitist algorithm. An alternative approach is to use a *roulette wheel selection* where a codebook is chosen into crossover with a probability that is an inverse function of its fitness value. Only the best codebook survives and all the rest are replaced by new codebooks produced by crossover operations. The weighting function (due to [4]) for a codebook is

$$f_i = \frac{1}{1 + MSE_i} \tag{3}$$

where MSE_i is the average distortion when the training vectors set has been coded by the vectors of the *i*'th codebook. The probability for a codebook to be chosen into crossover is:

$$p_i = \frac{f_i}{\sum_j f_j} \tag{4}$$

In order to compare these variants, let us next define the type of training vectors applied here. The training sets (and image types) considered are:

- 8 bits per pixel (bpp) image blocks taken from gray-scale images
- 1 bpp binarized blocks taken from gray-scale images
- 1 bpp image blocks taken from binary images

Training sets of the second type are derived by a BTC-like binarization, where the blocks are quantized into two levels according to the mean value of the pixels in the block [11, 12]. Here the binary blocks represent the *quantization matrix* (or *bit plane*) of the blocks. A codebook generated from the third type could be applied in *lossy* binary image compression. The training sets are summarized in Table 1, and the

originals of the gray-scale images are shown in Figure 3. The training set *ccitt* consists of the eight CCITT standard test images.

Bridge (256×256) Camera (256×256) Couple (256×256) Lena (512×512)

Figure 3. Training set images.

Table 1. Training sets and their statistics.

Training set	bpp	Original image	Preprocess	# vectors	# different vectors
BRIDGE	8	bridge	---	4096	4096
CAMERA	8	camera	---	4096	4095
COUPLE	8	couple	---	4096	3934
LENA	8	lena	---	16384	16384
BRIDGE-2	1	bridge	BTC-like binarization	4096	2813
CAMERA-2	1	camera	BTC-like binarization	4096	3172
COUPLE-2	1	couple	BTC-like binarization	4096	2119
CCITT	1	ccitt1..ccitt8	---	2052864	5987

The distortion values (between the training set and the codebook) of the main variants are shown in Table 2. The distortion values are the MSE-values for the gray-scale images (*bridge*, *couple*, *camera, lena*). For the binary images (*bridge2*, *couple2*, *camera2*, *ccitt*) the distortion is expressed by the average number of distorted pixels per 4×4-block (varying from 0 to 16). The CB-variant clearly outperforms the PB-variant in these tests. The results of the PB-variant do not even compare to a randomly chosen codebook (a codebook of 256 randomly chosen vectors from the training set). Of the two selection criteria the elitist version outperforms the roulette wheel selection in almost all cases. Mutations improved the results slightly.

Figure 4 illustrates the distortion values of the best codebook in each generation. Without mutations GA converges rather soon; 78 rounds is needed for *bridge*. The improvement of the first 25 iterations is 8.9 % whereas the improvement of the next 25 round is only 0.7 %. With mutations the situation is slightly different; the first 25 iterations improve the distortion values by 7.6 % whereas the next 25 round by 3.2 %. After 1000 generations extra improvement of 7.4 % (resulting to 209.2) were achieved (*bridge*), but at the cost of 20 times longer running time. These figures also demonstrate that the overall improvement remains rather small: 11 % for *bridge*, but only 5 % for *bridge2* (50 iterations).

The correlation between the generation size and the distortion is illustrated in Figure 5. A large generation size improves the result, but again, at the cost of

increased running time. Additional tests showed that if more computing time can be spent, the increase of iteration rounds has a slight edge over the increase of generation size. The difference, however, is quite small and basically the distortion of the best codebooks depends more on the number of candidate codebooks tested. In the rest of this paper the generation size is fixed to S=45.

Table 2. Distortion of the codebooks generated by the genetic algorithms (M=256, S=45, T=50, no mutations). Results for the random codebooks are average values of 50 test runs.

	CB-variant		PB-variant		Random
	Elitist	**Roulette**	**Elitist**	**Roulette**	**codebook**
BRIDGE	**228.92**	248.45	618.71	697.31	261.01
CAMERA	**160.97**	184.90	684.31	867.81	235.71
COUPLE	**49.72**	58.39	213.55	304.92	70.58
LENA	**83.03**	93.12	1004.42	1316.31	104.52
BRIDGE2	**1.46**	1.53	2.04	2.06	1.56
CAMERA2	**1.76**	1.80	2.22	2.32	1.84
COUPLE2	**1.06**	1.11	1.91	1.88	1.13
CCITT	**0.25**	**0.25**	1.67	1.77	0.27

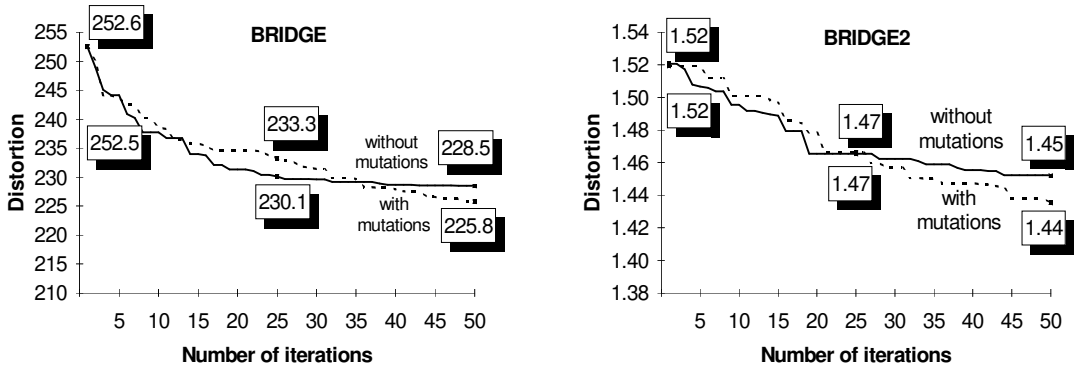

Figure 4. Number of iterations vs. distortion.

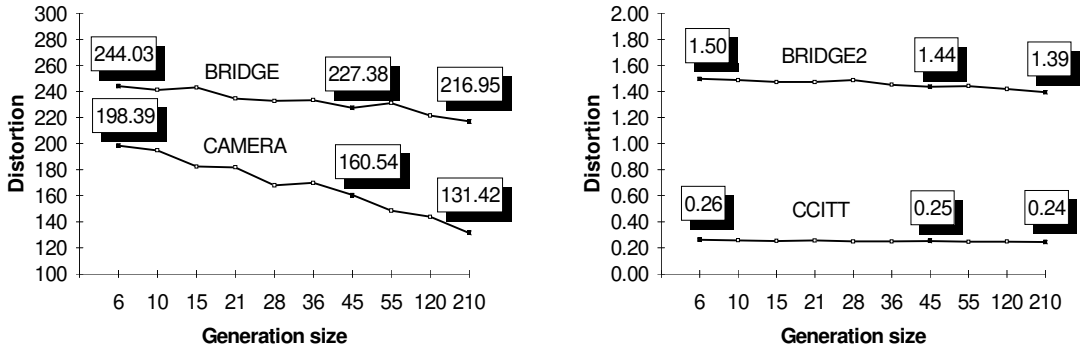

Figure 5. Generation size vs. distortion.

The computational complexity of the two variants is analyzed next. For each candidate solution (codebook or partitioning) we have to calculate both the partitions and the codevectors in order to calculate the fitness value of the solution. Thus, the CB- and PB-variants consist of all the same phases as GLA: *partitioning*, *centroid calculations*, and *MSE-calculations*. Their complexity in GLA is:

- Partitioning: $O(NM)$
- Centroid calculations: $O(N)$
- MSE-calculations: $O(N)$

In the CB-variant the centroids are calculated in the *crossover* operation. This takes $O(M)$ time consisting of M random picks from the two parent codebooks. Unfortunately the overall running time is still $O(NM)$ due to the partitioning. The PB-variant performs the time consuming partitioning in the crossover operation in $O(N)$ time. This is done by N random picks from the mapping function tables of the parent codebooks. Thus, the overall time complexity of the PB-variant is $O(N)$. Besides the previous operations, the genetic algorithms include also selection and mutations but their effect on the overall running time is negligible.

To sum up, the time complexity of the CB-variant is of the same order as that of GLA for a single codebook. There are, however, S codebooks (S=45 by default) to be generated in each iteration making the CB-variant much slower in overall. The PB-variant, on the other hand, has lower time complexity than GLA but this is also compensated by the large number of codebooks to be considered, see Table 3.

Table 3. Summary of the running times of the genetic algorithms. The example values have been calculated using the parameters N=4096, M=256, S=45, T=50.

	Algorithmic complexity			**Example values**		
	CB-variant	**PB-variant**	**GLA**	**CB-variant**	**PB-variant**	**GLA**
Partitioning	$O(NM)$	$O(N)$	$O(NM)$	10^6	4 096	10^6
Centroid calculations	$O(M)$	$O(N)$	$O(N)$	256	4 096	4 096
MSE-calculations	$O(N)$	$O(N)$	$O(N)$	4 096	4 096	4 096
Total (1 solution)	$O(NM)$	$O(N)$	$\boldsymbol{O(NM)}$	10^6	12 288	$\mathbf{10^6}$
Total (S solutions)	$\boldsymbol{O(SNM)}$	$\boldsymbol{O(SM)}$	---	$\mathbf{45 \cdot 10^6}$	**552 960**	---

The actual observed running times of the genetic algorithms are summarized in Table 4. All algorithms were implemented in C-language and the test runs were performed on a Pentium/90 computer. The number of iterations in GLA was set (21, 48, 4) for the training sets (*bridge*, *lena*, *ccitt*). In the genetic algorithms the number of iterations were set 50. No mutations were applied.

The actual running times correspond surprisingly well to the example values given in Table 3; for example, the times for the PB-variant and GLA (for one iteration) are of the same magnitude whereas the times for the CB-variant are clearly higher. The slow running time is clearly a drawback of the genetic algorithms, even when compared to

the GLA. The running time, on the other hand, can be adjusted by changing the generation size *S*, see Figure 6.

Table 4. Running times (*hours:minutes:seconds*) of GA and GLA.

	Time per iteration round			Total running time		
	CB-variant	PB-variant	GLA	CB-variant	PB-variant	GLA
BRIDGE	1:25	0:04	0:03	1:03:33	2:58	1:11
LENA	3:49	0:15	0:11	3:11:04	12:30	9:06
CCITT	2:13	0:05	0:08	2:13:13	4:32	0:33

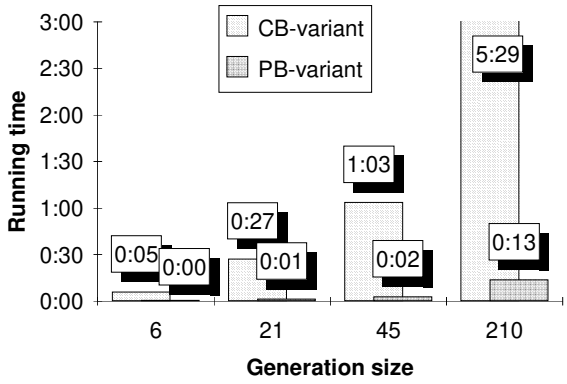

Figure 6. Generation size vs. running time (minutes) for *bridge* (*T*=50)

3.2 Crossover algorithms

The question of producing the new individuals in the crossover operation is studied next. The object of crossover is to create a new (and hopefully better) codebook on the basis of two given parent codebooks (denoted here by *A* and *B*). Since the codevectors are the elementary units of the individuals, the crossover can be simplified to a selection of *M* codevectors from the two parent codebooks. Let us next study two existing crossover algorithms and introduce three new methods:

- Random crossover [4]
- Centroid distance [5]
- Pairwise crossover
- Largest partitions
- PNN algorithm

Random crossover

A random *multipoint crossover* algorithm is implemented by picking *M*/2 randomly chosen codevectors from each of the two parent codebooks. Duplicate codevectors are rejected and replaced by repeated picks. This algorithm is extremely simple and quite efficient, because there is (in the unsorted case) no correlation between neighboring genes to be taken advantage of.

Centroid distance

If the codevectors in the codebook are sorted by some criterion a *single point crossover* is possible. In [5] the codevectors are sorted according to the distance between the codevector and the centroid of all codevectors in the codebook. In a sense, the codebook is divided into two subsets. The first subset (*central vectors*) consists of the codevectors that are close to the centroid, and the second one (*borderline vectors*) of the vectors that are far from the centroid. Now the new codebook is created by taking the central vectors from codebook *A* and the borderline vectors from codebook *B*. The cutting point can be anything between 1 to *M* but we use the halfpoint (*M*/2) in our implementation.

Pairwise crossover

It is desired that the new individual should inherit different genes from the two parents. The sorting of the codevectors was an attempt of this kind but it can be developed even further. By searching the nearest codevector (in the codebook *B*) for every codevector in the codebook *A* we can pair the codevectors between the two codebooks. The crossover is then performed by taking one codevector (by random choice) from every couple of vectors. By this way we can avoid taking similar codevectors from both of the parent codebooks. The nearest codevectors are found in a greedy manner for each vector in *A*, by taking the nearest available vector in *B*. A codevector that has already been paired cannot be chosen second time. The last codevector in *A* is paired with the only one left in *B*. This algorithm does not give the optimal *pairing* (*2-assignment*) but it is a reasonable good heuristic for the crossover purpose.

Largest partitions

In the *Largest partitions* algorithm the *M* vectors are picked by a greedy heuristic based on the assumption that the codevectors with larger partitions are more important than the vectors with smaller partitions. Each codevector in the codebooks *A* and *B* is assigned with a number, *partition size*, indicating how many training vectors is mapped to it. In each phase, the codevector with the largest partition size is picked. Assume that the codevector (denote by *X*) was chosen from *A*. The partition size of *X* is then set to zero in the codebook *A* to avoid reselection in the next round. In order to avoid similar (or even the same) codevector to be chosen from the other parent codebook, the partitions must be updated in *B* thoroughly. This is performed by removing the effect of those training vectors in *B* that were mapped to the chosen codevector *X* in *A*.

PNN algorithm

All the above crossover algorithms have the limitation that they cannot create new codevectors but are limited to those in the parent codebooks. This restriction, however, is only due to the *picking strategy*; where the object is to pick *M* vectors from the 2*M* codevectors of the parent codebooks. An alternative strategy is to

consider the crossover phase as a special case of the codebook construction problem itself. In fact, if we combine the codebooks A and B, their union can be considered as a training set of $2M$ vectors. Now the aim is to create a codebook of size M from this training set. This can be done by any existing codebook generation algorithm. Here we consider the use of *Pairwise Nearest Neighbor* (*PNN*) algorithm [13].

The PNN algorithm starts by initializing a codebook of size $2M$ where the codebook contains all the training vectors. Then two codevectors (partitions) are combined and replaced by the centroid of the training vectors belonging to the combined partitions. The vectors to be combined are the ones that increase the distortion least. This step is then repeated M times, after which the size of the codebook has decreased to M.

Other ideas

So far we have assumed that the starting point of the crossover is two parent codebooks. However, there is no real restriction to the number of codebooks in the crossover. In fact, we can even put the codevectors of all surviving codebooks into same set (denoted by *pool*) and then generate a set of new codebooks on the basis of this pool. For example, we can consider this pool as a training set and generate codebooks by any codebook generation algorithm. The only restriction is that the new codebooks should be different from each other in order to keep the evolution going on. The following algorithms might be considered:

- An algorithm containing randomness (eg. GLA with randomly chosen initial codebooks) so that the resulting codebooks will be different.
- A set of different algorithms (eg. GLA, PNN, Binary splitting)

The latter one is restricted by the fact that the number of the algorithms should be sufficiently large (cf. the size of the population, S=45, used in the present study). The pool strategy is not studied further in the present paper.

The results of the various crossover algorithms are summarized in Table 5. Here mutations have been applied with the probability P_m=0.01. The new crossover algorithms (*Largest partitions*, *Pairwise crossover*, *PNN*) outperform the existing ones (*Random crossover* and *Centroid distance*). The best algorithm seems to be *PNN* for the gray scale images and the *Largest partitions* method for the binary images.

The results of the deterministic (non-random) crossover algorithms are greatly effected by the use of mutations. Without mutations the *Centroid distance* algorithm, for example, converges always after two or three iterations rounds and the corresponding distortion values are much worse (251.43 for *bridge*) than with mutations (228.11). The *Pairwise crossover* and *Largest partitions* can keep the evolution going on long enough but the results are still much worse (228.77 and 236.58 without mutations) compared to the result of *random crossover* (the point of comparison), at least in the case of gray scale images. The *PNN algorithm*, on the other hand, is different from all other algorithms. It seems to converge very quickly (with or without mutations) taking at most 4 iteration rounds. Moreover, it suffers much less from the lack of mutations than the other crossover algorithms.

Table 5. Distortion of the codebooks using different crossover algorithms (S=45, T=50, mutations with P_m=0.01).

	Random crossover	**Centroid distance**	**Pairwise crossover**	**Largest partitions**	**PNN algorithm**
BRIDGE	225.04	228.11	223.41	**221.65**	223.12
CAMERA	155.58	162.17	149.13	158.49	**123.52**
COUPLE	48.64	48.65	46.46	47.57	**41.48**
LENA	81.59	84.36	82.25	80.16	**77.98**
BRIDGE-2	1.44	1.43	1.40	**1.35**	1.51
CAMERA-2	1.72	1.71	1.68	**1.61**	1.78
COUPLE-2	1.06	1.05	1.01	**0.97**	1.11
CCITT	0.25	0.24	**0.22**	**0.22**	0.25

4. Integration with GLA

GLA and GA are similar in the sense that they both need an initial codebook (or population of codebooks) to start with. This property can be utilized by integrating the algorithms. There are three obvious ways to implement this:

- Apply GLA for the output of GA (*GA+GLA*).
- Apply GA for the output of GLA (*GLA+GA*)
- Apply GLA for every codebook in each GA population (*Hybrid*)

The first one (GA+GLA) is a straightforward improvement of the basic GA algorithm. Whatever is the output by GA, the best codebook of the last generation can always be iterated by GLA resulting to a better or equal codebook.

The second alternative (GLA+GA) is not so obvious. The problem is that GA needs several codebooks to start with (45 by default in this paper), but GLA produces only one. It is desirable that the initial codebooks would be different from each other but this property is not obligatory. Thus, we could take 45 copies of the same codebook, but a better choice is to repeat the GLA algorithm for 45 different (randomly chosen) initial codebooks resulting 45 different GLA codebooks. Experiments showed that without mutations, the proposed crossover techniques could not produce any better codebooks. With mutations, moderate improvements were achieved. Note that the starting point of GA does not need to be generated by GLA but any other algorithm is applicable; the use of *binary splitting* method was proposed in [4].

The third alternative (Hybrid) is to integrate GA and GLA so that every codebook in each population is iterated by GLA. This scheme was proposed in [5], but also the PB-variant proposed in [4] includes (hiddenly) one and a half iterations of GLA. By this way the search space can be reduced by eliminating all other solutions but those that are local minima (as defined in GLA). The main drawback of the *hybrid* algorithm is the increased running time (approximately multiplied by the number of GLA-iterations).

Figure 7 illustrates how the result of the hybrid algorithm changes when the number of GLA-iterations increases. It seems that the integration of GLA itself is important, but the number of iterations is of secondary importance. For example, the distortion values for *bridge* are (175.2, 170.6, 169.8) when applying (1, 2, and 20) iterations of GLA in each generation. Compare this to the value 225.0 when no GLA was applied. In the further tests the number of GLA-iterations will be fixed to 2.

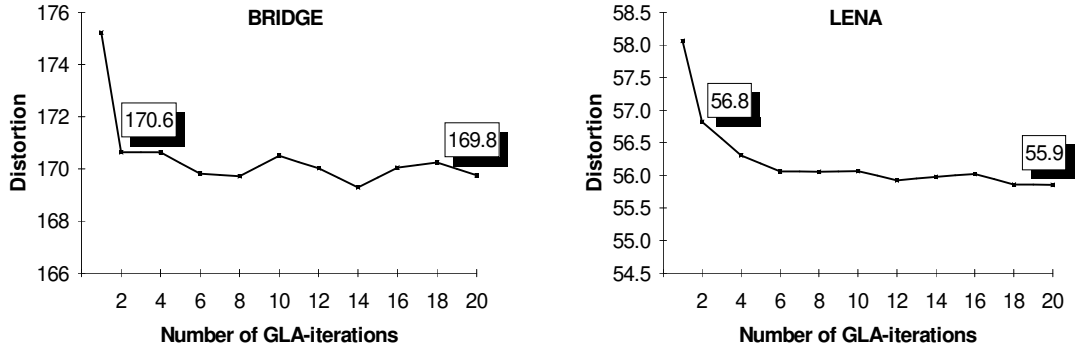

Figure 7. Distortion of the hybrid codebooks with various amount of GLA-iterations. (S=45, T=25, mutations with P_m=0.01)

Table 6 gives test results for the hybrid algorithm with different crossover algorithms. The results show that there is no prime candidate for the crossover algorithm anymore. For example, the *Pairwise crossover* is the best for three gray scale images out of four, but only with a small marginal. In the case of binary images, the *Largest partitions* method has a slight edge over the others (excluding *ccitt*). The *Random crossover*, on the other hand, never gives the best result but is always very close; thus having no weak points among the training sets. It was noted, that the hybrid algorithm always converged rather soon; thus only 25 iterations was applied.

Table 6. Distortion of the hybrid codebooks using various crossover algorithms (S=45, T=25, mutations with P_m=0.01).

	Random crossover	**Centroid distance**	**Pairwise crossover**	**Largest partitions**	**PNN algorithm**
BRIDGE	172.02	171.21	**167.72**	171.68	173.00
CAMERA	78.96	91.65	84.09	102.20	**77.45**
COUPLE	28.49	30.92	**27.83**	32.35	27.95
LENA	57.06	57.07	**56.35**	57.73	57.65
BRIDGE-2	1.35	1.33	1.34	**1.31**	1.44
CAMERA-2	1.63	1.60	1.61	**1.58**	1.71
COUPLE-2	0.97	0.95	0.97	**0.94**	1.01
CCITT	0.21	**0.20**	0.23	0.21	0.22

Table 7 summarizes the best results obtained by genetic algorithms. The best particular crossover method (see Tables 5 and 6) is applied to each image. For comparison, results for the following algorithms have been included:

GA = Genetic algorithm.
GLA = Generalized Lloyd Algorithm [2].
GA+GLA = GLA for the output of GA.
GLA+GA = GA for an initial population of 45 different GLA codebooks.
Hybrid = Hybrid of GA and GLA.
SA = Simulated Annealing [8].
PNN = Pairwise Nearest Neighbor algorithm [13].

PNN algorithm was implemented so that the combined codevectors were optimally chosen, i.e. taking the ones that increase the distortion least when combined. In the simulated annealing, a random noise vector was applied with a uniform *pdf* in the range [-*t*, *t*]. A logarithmic temperature schedule decreased the temperature *t* by 1 % after each iteration step. The initial temperature t_0 was 50 for gray-scale images, and 1 for binary images. The inclusion of noise was stopped when *t* fell below 0.5; smaller temperature values have no effect because the pixel values are always rounded to nearest integer.

The results in Table 7 show that the pure GA gives better results than GLA for binary images but cannot compete in the case of gray scale images. For example, the distortion value for *bridge* is 221.65 for the best combination of GA, and 179.68 for pure GLA. The corresponding results for SA and PNN are 162.45 and 169.15. The results of the GA+GLA and GLA+GA combinations are much better than those of pure GA, but still not as good as those of SA and PNN. The results of the hybrid combination of GA and GLA are close to those of SA and PNN (167.72 for *bridge*).

Table 7. Performance comparisons of various algorithms (distortion values).

	GA	**GLA**	**GLA+GA**	**GA+GLA**	**Hybrid**	**SA**	**PNN**
BRIDGE	221.65	179.68	177.61	175.82	167.72	**162.45**	169.15
CAMERA	123.52	122.61	111.61	97.11	77.45	**70.65**	70.90
COUPLE	41.48	40.36	35.73	30.98	27.83	**25.15**	25.91
LENA	77.98	59.73	58.89	57.69	56.35	**54.40**	56.44
BRIDGE-2	1.35	1.48	1.32	1.41	**1.31**	1.52	1.33
CAMERA-2	1.61	1.76	1.60	1.60	**1.58**	1.78	1.61
COUPLE-2	0.97	1.06	0.95	**0.94**	**0.94**	1.12	0.95
CCITT	0.22	0.24	0.26	0.21	0.20	0.40	**0.18**

Let us next compare the efficiency of the algorithms. We calculate the distortion of the best codebook as a function of the running time spent. Since GLA is much faster than GA and the hybrid algorithm, the extra time can be spent by repeating GLA, each time starting from a new random codebook. 400 runs were performed in total during an 8 hour time period with GLA. The variation of the GLA results, however, is too small to achieve any remarkable improvement when repeated, see Figure 8. Simulated annealing took 25 minutes and was similarly repeated 18 times with similar results.

GA continues to improve the result, but it is unlikely that the results would ever reach the same level as those of Hybrid and SA, see Figure 8. The hybrid algorithm converged after 44 iterations (11 hours) without reaching the level of SA. Thus, in the

case of gray scale images SA is clearly superior to the variants of genetic algorithm, both in time and distortion. However, in the case of binary images SA doesn't seem to work, resulting in the highest distortion values for all images. In this case, the best method seems to be the Hybrid algorithm, unless the running time is a critical factor, in which case GLA becomes an attractive alternatives. Note, that the running time of PNN highly depends on the size of the training set. For example, our implementation of PNN took 67 minutes for *bridge* (4096 training vectors), but 18½ hours for *lena* (16384 vectors).

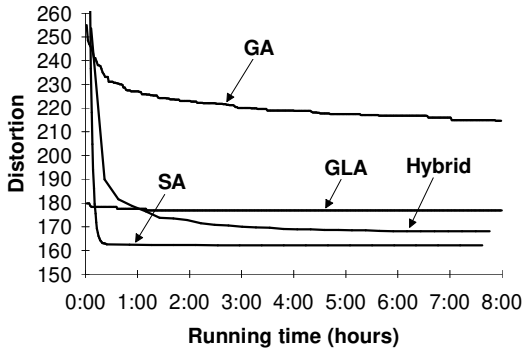

Figure 8. The distortion of the best codebook found as a function of the execution time (for *bridge*).

5. Conclusions

GA solutions for the codebook generation problem of VQ were studied. The problem of the partition-based variant is that the clusters are practically random and thus the resulting codevectors are concentrated near the centroid of the training set. This drawback could be eliminated if the fitness function was calculated on the basis of the resulting codebook instead of the partitioning. This modification, however, means that the (computationally expensive) nearest neighbor partitioning should be applied; and thus the speed advantage of the partition-based variant would be lost.

The experiments showed that a pure genetic algorithm is competitive with the existing methods only for binary images. For gray scale images, the inclusion of GLA iterations is vital. Even now, the time-distortion performance of the hybrid algorithm is inferior to simulated annealing. For binary images, the hybrid combination of GA and GLA gives the best results. The greatest deficiency of the algorithm is its high running time. If time is a critical factor, GLA becomes a more attractive alternative.

The choice of the crossover algorithm seems to be of secondary importance, and highly depend on the type of training set. The main problem of (otherwise good) deterministic crossover algorithms, with the combination of GLA iterations, is that together they reduce the amount of randomness, which is vital in genetic algorithms. Moreover, the main benefit of the algorithm seems not to be generating the best possible codebook, but to find a better initial codebook for GLA.

One must keep in mind that the randomness makes the results somewhat unpredictable, cf. the peaks in Figure 7. Therefore, the results would be more reliable if they were calculated from the averages of several test runs. Some of the experiments indeed were repeated with consistent results, but it would be inconvenient to repeat the whole process several times because of the huge computing efforts required by the algorithms.

References

[1] **M.R. Garey, D.S. Johnson, H.S. Witsenhausen,** The Complexity of the Generalized Lloyd-Max Problem. *IEEE Transactions on Information Theory*, Vol.28 (2), pp.255-256, March 1980.

[2] **Y. Linde, A. Buzo and R.M. Gray,** An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, Vol.28 (1), pp.84-95, January 1980.

[3] **D.E. Goldberg**, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading (1989).

[4] **V. Delport and M. Koschorreck**, Genetic Algorithm for Codebook Design in Vector Quantization. *Electronics Letters*, Vol. 31, (2), pp. 84-85, January 1995.

[5] **J.S. Pan, F.R. McInnes and M.A. Jack**, VQ Codebook Design Using Genetic Algorithms. *Electronics Letters*, Vol. 31, (17), pp. 1418-1419, August 1995.

[6] **P. Fränti, T. Kaukoranta and O. Nevalainen,** Blockwise Distortion Measure in Image Compression. *Proc. Very High Resolution and Quality Imaging*, San Jose, CA, SPIE Vol. 2663, pp. 78-87, 1996.

[7] **A. Gersho, R.M. Gray,** *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[8] **K. Zeger and A. Gersho,** Stochastic Relaxation Algorithm for Improved Vector Quantiser Design. *Electronics Letters*, Vol. 25 (14), pp. 896-898, July 1989.

[9] **J. Vaisey and A. Gersho,** Simulated Annealing and Codebook Design. *Proc. ICASSP*, 1176-1179, April 1988.

[10] **K. Al-Sultan,** A Tabu Search Approach to the Clustering Problem. *Pattern Recognition*, Vol. 28 (9), pp. 1443-1451, 1995.

[11] **P. Fränti, T. Kaukoranta and O. Nevalainen**, On the Design of a Hierarchical BTC-VQ Compression System. *Signal Processing: Image Communication*, Vol. 8 (6), September 1996. (to appear)

[12] **P. Fränti, O. Nevalainen and T. Kaukoranta**, Compression of Digital Images by Block Truncation Coding: A Survey. *The Computer Journal*, Vol. 37 (4), pp. 308-332, 1994.

[13] **W.H. Equitz**, A New Vector Quantization Clustering Algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, (10), pp. 1568-1575, October 1989.