# A Fast Evolutionary Algorithm in Codebook Design

ABDOLALI MOMENAI, SIAMAK TALEBI
Electrical Engineering Department
Shahid Bahonar University of Kerman
Kerman
IRAN

*Abstract:* This paper presents a fast algorithm to find an optimal subset codebook from a super codebook in a way that the RMS error between the new codebook and the training set in vector quantization becomes minimum. To have a fast algorithm, a genetic based algorithm is used that uses 2 evolutions, one in designing the whole sub-codebook and the other in finding each individual codeword of the sub-codebook. This optimal codebook can be efficiently used in the real time compression of the images with PSNR improvement of about 1.1 – 8.2db in blocks of the image.

*Key-Words:* Vector Quantization, Image Compression, Adaptive VQ, Codebook Design, Genetic Algorithm

## 1 Introduction

### 1.1 Vector Quantization

Vector quantization (VQ) is a generalization of scalar quantization [1]. VQ divides the whole vector space into a limited number of vectors called codewords, and then quantizes each vector in the source with the closest codeword in the codebook and sends the index of the codeword in the codebook instead of the vector. It can achieve high compression rate with high quality if the codebook matches the source i.e. the error of quantization is minimal. It's obvious that every source has its own statistical parameters in the vectors that it produces. As a result for each different source, a different codebook should be used and the codebooks should be designed according to each specific source. Common codebook design algorithms use a number of sample vectors (training set) from the source to design a codebook that consists of a specific number of codewords. Unfortunately, there is no practical algorithm that leads to the optimal codebook for a given training sequence [1].

Current algorithms use the whole vector space as the source for choosing the codewords. By searching vectors close to the training sets, they finally find a sub-optimal codebook, such as Generalized Max-Lloyd Algorithm (GLA) [1], Linde-Buzo-Gray (LBG) [2], Pairwise Nearest Neighbor (PNN) [1], [3] and other methods. This way of searching involves lots of computation and a high complexity so that they are inefficient to design a new codebook for every new source. So they have to use one codebook in different sources which results in less performance.

As mentioned, VQ can be used in many different fields to compress the source data very effectively [4], [5], such as speech [6] and image [7]. However this paper focuses on the use of VQ in image compression and design of image codebooks. But it can be implemented in other fields with slight modification. In image compression, the image is divided into blocks of a specific size and each block is treated as a vector. For example, the image is divided into 4*4 blocks, i.e. the vector size is 16. Then by using a codebook of 256 codewords and finding the matching codeword from the codebook, the index of the codeword in the codebook is sent instead of the whole block. This results in image compression by a factor of 16 in 8 bit per pixel images. The decoder only should do a simple table lookup to replace the index with the codeword and reconstruct the image.

### 1.2 Genetic Algorithm

Nowadays, genetic algorithms (GA) [8], [9] and other evolutionary algorithms are widely used in finding solutions to different kinds of problems [10]. These evolutionary algorithms begin with a set of solutions to the problem which is called Initial Population. Then they try to make an improved population of new solutions by combining (Crossover) and modifying (Mutation) the solutions in the current population. Combining and modifying the solutions in each population are carried out in a way that is hoped to make a better solution. After sufficient number of iterations, if the algorithm is devised correctly, it will converge to a good solution which may be the optimal solution. The probability of finding the optimal solution or even a sub-optimal solution and the speed

of convergence, is directly related to Crossover, Mutation, and the other parameters of the algorithm.
Although these algorithms are popular, they are slow algorithms, because of the high number of iterations that are used in genetic algorithms or the other similar searching algorithms. As a result when high speed is needed they cannot be used efficiently. Doing accurate analysis of the algorithm according to the problem and using some preprocessing, some adjustments can be found that improve the performance so much that it can be even used in real time computations.

### 1.3 New Approach in Codebook Design

In this paper, the problem of designing codebooks in a limited time and limited calculations is solved by using a preprocessing: designing a Super Codebook. A super codebook is a very large codebook that consists of many codewords so that it can be used in many different sources. Then this super codebook is used in designing other codebooks in a way that they will be subsets of the super codebook. In this study, a super codebook of 8192 codewords by using GLA method is designed and the vector size is 16 i.e. image block size is 4*4. A very large training set of more than one million vectors is used to guarantee a sufficient statistical dependence of the super codebook from the training set. As mentioned, to design a codebook for a new source, the vector space of this super codebook is used instead of the whole vector space. Genetic algorithms can help finding a codebook subset of the super codebook that is optimal with respect to the new source. As shown in previous works, having a super codebook and using its subsets for each part of the image can result in a higher performance [7]. In Finite State Vector Quantization algorithms static subset codebooks are designed by using complex and slow algorithms and used in the quantization process [11]. Because these subsets are static, they are not optimal for every new source, so new subsets should be designed for each new source using a fast algorithm. In Adaptive Vector Quantization algorithms the sub-codebooks change through the quantization process according to some statistical parameters of the source [7]. But they have usually used a very simple analysis of the statistical parameters to have a fast algorithm. So the sub-codebooks can still be improved by more heuristic methods that are used in the proposed algorithm. As a result, one of the fields that the proposed algorithm can be used is when there is a source that has a high number of vectors. Using this algorithm and changing the sub-codebooks dynamically over the quantization process can result in a high performance gain. This paper mainly discusses the designing of a subset codebook according to each region of an image and proves its quality which will be more than the ordinary codebooks of the same size. Because of the detailed explanation of the algorithm, its further application in image compression and other different fields will be covered in future works.

## 2 GSCD Algorithm

### 2.1 Overview

In this section the proposed Genetic Sub-Codebook Design algorithm (GSCD) is discussed in detail. The flow chart of the algorithm is presented in Fig. 4. Different parts of the algorithm are described in the following subsections.

### 2.2 Algorithm Inputs

The inputs of the algorithm are a training set, a super codebook and a neighborhood list for each codeword in the super codebook. The neighborhood lists are previously built by another program that calculates the RMS distance between each pair of the codewords in the super codebook. Then it stores the closest neighbors of each codeword in a sorted list. In this study, the neighborhood list is limited to the size of about one hundredth of the super codebook size. The super codebook size is 8192 and 80 closest codewords are used as the neighborhood list of each codeword.

### 2.3 Solution Representation

The solutions of this algorithm are subsets of the super codebook. So any solution can be represented by a list that shows the indices of the codewords of the super codebook that are used in the subset.
In every population, stronger individuals live longer. This is a basic principle in evolutionary algorithms that is called Elitism [12]. The GSCD uses this principle: The top solutions of each population are directly passed to the next population. As a result they act as the leader of the population to the optimal solution.
The population size and the top solutions size, play an important role in converging the algorithm and the computational complexity. If a small population size is chosen, there is a risk of under-covering the solution space, while large population size needs more computation [13], [14]. After some experiments, the population size was chosen three times the sub-codebook size. The top solutions that go directly to the next population are one tenth of the whole population. For example, if the sub-codebook size is 32, then the population size is 96 (~100) and the top solutions are ten.

### 2.4 Main Fitness Function

The Main Fitness Function in this algorithm is the objective function which maximizes the Peak Signal to Noise Ratio (PSNR) of the sub-codebook with respect to the training sequence. To calculate this function, the closest codeword in the sub-codebook is found for each of the training set vectors. The search for the closest codeword is constrained using the annulus constrained method that uses substantially less computation than the full search method with no loss of accuracy. The RMS error of this codeword to the training sequence vector is calculated. This procedure is repeated for all of the training sequence vectors. The average of the RMS errors of all vectors and then the PSNR function is calculated. The frequency of usage (rate of occurrence) for each codeword in the training sequence is kept for further analysis.

### 2.5 Initial Population

Initial population plays an important role in every genetic algorithm. If more time is spent on making a good Initial population, it will help the algorithm converge faster. In several methods, some solutions that are calculated with different approaches were used as the starting point for the algorithm, and GA is used to improve those solutions [15], [16]. But there is a risk of loss of diversity in the initial population and the problem of being caught in the local optimal solutions. Experimental results show that a completely random initial population would result in a faster convergence, i.e. all the 100 initial sub-codebooks are chosen randomly from the super codebook.

### 2.6 Parent Selection

As mentioned above, there are 100 solutions in each population, ten of them are directly transferred to the next population and the remaining 90 solutions are made by combining current solutions. For combining the solutions, two solutions are selected as the parents and then combined together (Crossover) to make two children out of them. Since every solution is not suitable to be a parent for the next population, a good strategy should be implemented in selecting the good parents so that they result in good children in the next population. One of the parent selection strategies is Roulette Wheel Selection [9]. In this approach, the probability of a solution to be selected as a parent is directly proportional to the Fitness function in a way that good solutions have higher chance of being selected as parents. So 45 pairs of parents are chosen by using the Roulette Wheel selection algorithm and combined to make the next population.

### 2.7 Crossover

After selecting parents, a method is implemented to combine (Crossover) them and make two children out of them. The combination method is dependent on the problem and the constraints of the problem should be considered in combining the parents, so they result in good children. In this algorithm, two subsets of a large set are combined to form two other subsets. The way these 2 subsets are combined is a new heuristic method that will be discussed in the following subsections.

#### 2.7.1 Secondary Population

There is one primary population of sub-codebooks that evolves through the iterations of the algorithm and converges to the optimal solution. A secondary population is introduced here, the codewords of the super codebook. This secondary population evolves through the iterations of the algorithm and refines itself by putting the most useful codewords in higher ranks and deleting worst codewords from the population (Locking a Codeword).

#### 2.7.2 Secondary Fitness Function

A secondary fitness function for the secondary population is defined here. If each codeword of the super codebook is analyzed individually, it is found that each codeword plays a role in increasing the main fitness function (PSNR). The effect of a codeword on the PSNR is proportional to its frequency of usage in the training sequence. A codeword that is used more often in quantizing is better than a codeword that is used less often. As a result, the secondary fitness function is defined to be the frequency of codeword usage which is calculated at the end of section 2.4. This secondary fitness function is used for selecting the codewords from the parents to make children.

#### 2.7.3 Locking a Codeword

After calculating the secondary fitness function (PSNR), another procedure is implemented called locking a codeword. The codeword frequency for each of the codewords in the sub-codebook is analyzed. If it is lower than a predefined threshold, that codeword in the super codebook is locked and can not be used anymore in any solution for a predefined number of iterations. This procedure is the main part of the evolution of the secondary population, i.e. codewords of the super codebook. It helps the algorithm dramatically converge faster, by minimizing the number of useful codewords that the algorithm can choose from for making a sub-codebook.

### 2.7.4 Codewords Selection

Roulette Wheel is used again for selecting the codewords of the children sub-codebooks in the main part of the crossover algorithm. Codewords are selected from the two parents' codewords with a probability that is proportional to the secondary fitness function. Note that the locked codewords have the probability of zero until their lock period is passed and they become unlocked. So the above procedure is repeated until the two children grow to the size of a normal sub-codebook. But there might be a problem of choosing a codeword more than once in the children's subsets. Dealing with repeated codewords in a sub-codebook is discussed in the Repairing algorithm that follows the Mutation.

### 2.8 Mutation

Mutation is a key point in Genetic Algorithms that helps them get out from the local optimums [9]. As seen in the human being, the mutation is carried out naturally in each new generation so that the human being will adapt to the new situations. But some mutations may lead to a worse solution. This solution will not succeed to the next generation that follows this generation, by the help of the Roulette Wheel selection (Section 2.6). If a solution has a low fitness function, it will not be selected by the Roulette Wheel algorithm because the probability of the selection is proportional to the fitness function.

This idea is implemented in the algorithm in a way that each codeword in a sub-codebook has a very little chance of mutation. If the mutation happens, the codeword will change to one of its neighborhood codewords that are in the pre-calculated neighborhood list. The mutation rate should not be high to harm the evolution process. According to the experiments on this algorithm, the size of the mutation neighborhood is limited to the first ten closest codewords, i.e. the first ten codewords in the pre-calculated neighborhood list.

### 2.9 Repairing Unfeasible Solutions

The last part of the algorithm which is significantly efficient in the convergence of the algorithm repairs the solutions that are not feasible. An unfeasible solution to the problem is a sub-codebook that contains a codeword more than once, which can occur in the crossover or mutation algorithms. The first occurrence of any codeword is kept in the sub-codebook and duplicated codewords are changed to new codewords that are not in the sub-codebook. When a repeated codeword is found, it is changed to one of its neighborhood codewords and if all of the neighborhood codewords are in the sub-codebook, the neighborhoods of its neighborhood vectors are searched and so on, until a codeword that is not in the sub-codebook is found. Studies show to search the neighborhood list of a codeword, it's better to search it randomly and not to start from the first codeword to the last codeword in the list (the list is sorted from the closest neighbors to the farther neighbors). In the study, the size of neighborhood list for the repairing algorithm is the whole precalculated neighborhood list of the vectors.

### 2.10 Stopping Condition

Stopping condition depends on the time allocated to the algorithm to search for the solutions. It can stop after the difference between the old population and the new population is smaller than a given threshold. However Evolutionary algorithms may not improve solutions for some iterations, but suddenly they improve the solutions. So it's better to consider the behavior of the algorithm for a number of iterations, and if it can't improve the solutions after a predefined number of iterations, the algorithm is stopped. It can also stop after a constant number of iterations. After some experiments and analyzing the convergence of the algorithm, the number of evolutions is limited to 15 iterations. Because the algorithm almost converges in 15 iterations and more iterations won't yield any significant improvement.

## 3 Experimental Results

To show the efficiency of the GSCD algorithm, the Lena image with 512*512 pixels (16384 vectors) is divided into 64 blocks of 64*64 pixels (each block has 256 vectors). Then the best sub-codebook with 16 vectors of the super codebook for each block of the image is found by using each block as the training set for the GSCD algorithm. So there are 64 blocks of the image with 64 sub-codebooks, one sub-codebook for each block. Then each block is encoded with three different codebooks and the PSNR is calculated. The first codebook is the super codebook with 8192 vectors. The second codebook (small codebook) has 16 vectors designed by using GLA algorithm from the same training sequence that the super codebook is designed from. The third codebook is the sub-codebook generated by using GSCD and has 16 vectors. The bitrate for each of these three methods is calculated (Table 1). For GSCD, all sub-codebooks should be sent to the decoder so it can decode the image. So it yields more bitrate than the usual 16 vector codebooks. In this study all indexes of the sub-codebooks from the super codebook are sent to the decoder which results in 0.8 more bits for each block of the images. To decrease the overhead of sending these sub-codebooks to the decoder some methods can

be implemented. For example the difference of each sub-codebook with respect to the previous sub-codebook can be sent or even some heuristic methods can be used.

As you see in Fig. 5, in those blocks of the image with less detail, PSNR has a higher improvement up to 8.2db and in detailed blocks, the improvement is low. The size of the sub-codebooks can be changed dynamically along the image i.e. larger sub-codebooks are used in those blocks with more details and smaller ones in less details so it yields higher PSNR in almost the same bitrate. Some problems such as sizing each block of the image, and the sub-codebook for each block and transmitting sub-codebooks to the decoder efficiently are not discussed here and will have their detailed implementation in our future works.

# 4 Conclusion

Using this fast algorithm that converges in limited number of iterations, the vector space of a super codebook is searched thoroughly to design a sub-codebook. As GSCD suggests, it's better to use a small sub-codebook for small blocks of data instead of using a large static codebook for all vectors of the source. By dynamically changing the block size, the sub-codebook size and even some parameters in the algorithm, high improvement can be reached with acceptable complexity. As it is dependent from the nature of the source, GSCD can be used in different areas that vector quantization is used such as speech, image and video compression. Implementation of a practical encoder and it's dynamically changing behavior will be discussed in our future works.

References:


[1] A. Gersho, R. Gray. "*Vector Quantization and Signal Compression*", Kluwer Academic Publishers, (1992)

[2] Y. Linde, A. Buzo, R. M. Gray. "An Algorithm for Vector Quantizer Design", *IEEE Trans. Commun.*, Vol. COM-28, pp. 84-95, January (1980)

[3] W.H. Equitz. "*Fast Algorithms for Vector Quantization Picture Coding*", Master's thesis, M.I.T., Cambridge, June (1984)

[4] R. M. Gray. "Vector Quantization", *IEEE ASSP Magazine*, pp. 4-29, April (1984)

[5] A. Gersho. "On the Structure of Vector Quantizers", *IEEE Trans. Inform. Theory*, Vol. IT-28, pp. 4-29, March (1982)

[6] J. Makhoul, S. Roucos, H. Gish. "Vector Quantization in Speech Coding", *Proc. IEEE*, vol. 73, pp. 1551-1588, November (1985)

[7] N. M. Nasrabadi, R. A. King. "Image Coding Using Vector Quantization, A review", *IEEE Trans. Commun.*, Vol. 36, pp. 957-971, August (1988)

[8] J. H. Holland. "*Adaptation in Natural and Artificial Systems*", University of Michigan Press, (1975)

[9] D. E. Goldberg. "Genetic Agorithms in Search, Optimization, and Machine Learning", Addison-Wesly, (1989)

[10] C.R. Reeves. "Modern Heuristic Techniques for Combinatorial Problems", McGraw-Hill, (1995)

[11] R. F. Chang, W. T. Chen. "Image Coding Using Variable-Rate Side-Match Finite-State Vector Quantization", *IEEE Trans. Image Processing*, Vol. 2, No. 1, pp. 104-108, January (1993)

[12] K. A. De Jong. "*An Analysis of the Behavior of a Class of Genetic Adaptive Systems*", Doctoral Dissertation, University of Michigan, (1975)

[13] D. E. Goldberg. "Sizing Populations for Serial and Parallel Genetic Algorithms", *Proceedings of 3rd International Conference on Genetic Algorithms*, (1989)

[14] J. T. Alander. "Optimal Population Size of Genetic Algorithms", *Proc. CompEuro 92, IEEE Computer Society Press*, pp. 65-70, (1992)

[15] C. R. Reeves. "A Genetic Algorithm for Flowshop Sequencing. Computers", Ops. Res, (1992)

[16] A. Kapsalis, G. D. Smith, V. J. Rayward-Smith. "*Solving the Graphical Steiner Sree Problem Using Genetic Algorithms*", JORS, (1993)


**Table 1** PSNR and Bitrate/Vector for three methods. Vector size is 4*4 = 16 pixels

| | Super CB | Small CB | GSCD |
|---|---|---|---|
| PSNR (db) | 33.70 | 26.06 | 28.20 |
| Bitrate/Vector | 13 | 4 | 4.8125 |

**Fig. 1.** GSCD

**Fig. 2.** Small Codebook

**Fig. 3.** Super Codebook

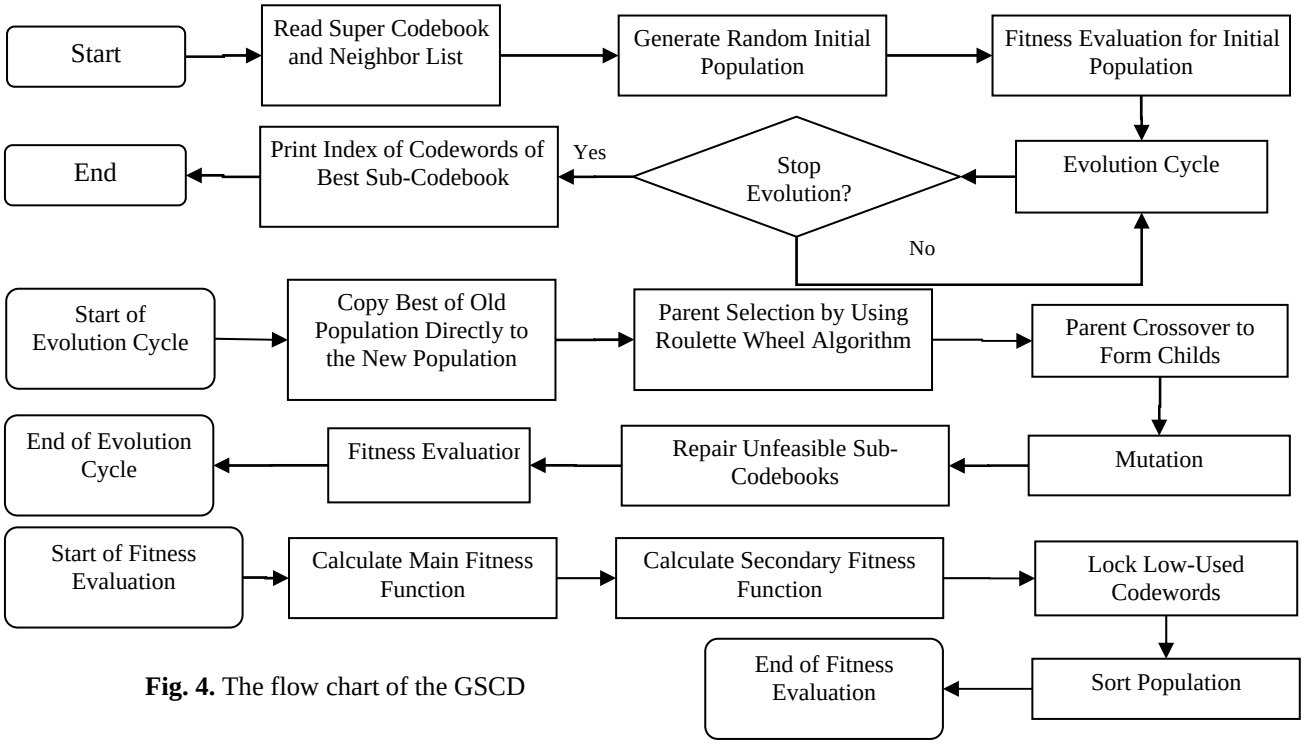


**Fig. 4.** The flow chart of the GSCD

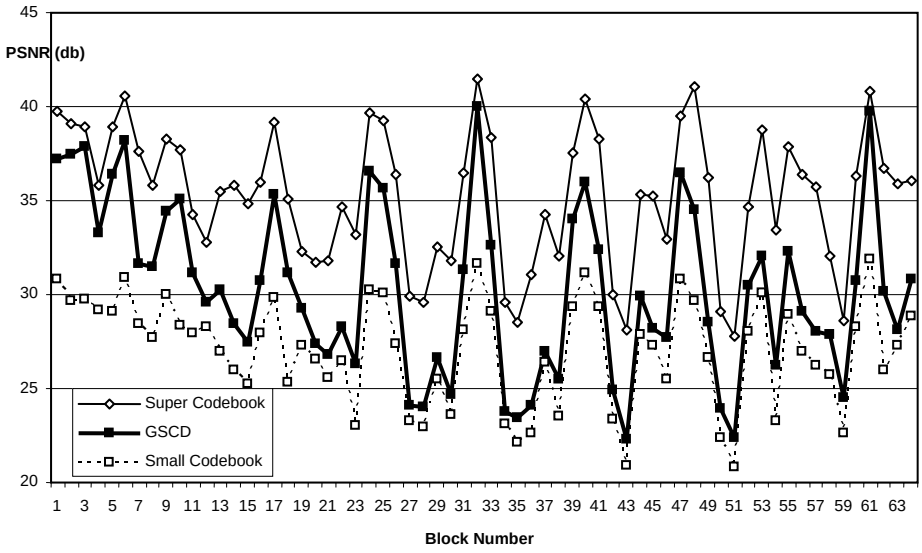


**Fig. 5.** Performance Comparison