Sound Data Compression Method Using Genetic Algorithm[1]

Mohammed A. F. Al-Husainy [2]

Abstract

The principal objective of this research is an adoption of the Genetic Algorithm (GA) for studying it firstly, and to stop over the operations which are introduced from the genetic algorithm.The candidate field for applying the operations of the genetic algorithm is the sound data compression field. This research uses the operations of the genetic algorithm for the enhancement of the performance of one of the popular compression method. Vector Quantization (VQ) method is selected in this work. After studying this method, new proposed algorithm for mixing the (GA) with this method was constructed and then the required programs for testing this algorithm was written. A good enhancement was recorded for the performance of the (VQ) method when mixed with the (GA). The proposed algorithm was tested by applying it on some sound data files. Some fidelity measures are calculated to evaluate the performance of the new proposed algorithm.

[1] For the paper in Arabic see Pages (33-34).

[2] Department of Computer Science, Faculty of Sciences and Information Technology, Al-Zaytoonah Private University of Jordan

1. Introduction

Firstly, we note that through the context of this paper, we refered to the sound signal as a data, because its treated as a set of data values. Vector quantization (VQ) has been widely used for data compression due to its theoretical advantages compared with scalar quantization schemes. However, the computational complexity of both the codebook design and the vector lookup during the encoding phase is obviously a burden of its realization. Since the codebook can be pre-generated before the encoding process, the efficiency of the vector lookup is comparatively more significant [Chan]. Vector Quantization (VQ) has been successfully used in speech and image data compression [Liang].

One of the emerging technologies for lossy data (sound data in this work) compression is Vector Quantization (VQ). Vector Quantization can be used to take advantage of the correlation between neighboring values of the signal (i.e., in the sound signal) by quantizing these values in groups (or vectors) rather than individually and symbolically representing the vector with a codeword. The appropriate codeword is chosen from the available codebook by minimizing a given distortion measure. Often, the most common distortion measure, between vectors of the codebook, is the mean-square error (MSE):

$$d_2(\vec{x},\vec{y}) = \frac{1}{N}\sum_{k=0}^{N-1}(\vec{x}_k - \vec{y}_k)^2$$

where the distortion is defined per dimension. The popularity of the (MSE) lies mainly in its simplicity and mathematical tractability. A more general distorton measure based on the L_r norm is defined by:

$$d_r(\vec{x},\vec{y}) = \frac{1}{N}\sum_{k=0}^{N-1}\left|\vec{x}_k - \vec{y}_k\right|^r$$

Note that d_r is equal to d_2 for $r = 2$. The other most popular values of r are r=1 and $r = \infty$. d_1 represent the average absolute error and d_∞ tends towards the maximum error [Cabral].

Vector quantization (VQ) is a source coding methodology with a provable rate-distortion optimality. However, despite more than two decades of intensive research, VQ's theoretical promise is yet to be fully realized in image compression practice [Xiaolin]. Data compression using

vector quantization (VQ) has received great attention in the last decade of its promising compression ratio and relatively simple structure. In its simplest implementation, VQ requires divide the signal, to be compressed, into *vectors* (which may be referred to as a *blocks*). Each vector of the signal to be compressed is compared to the entries of a *codebook* containing representative vectors. The *address* of the codebook entry most similar to the signal vector is then transmitted to the receiver, where it is used to fetch the same entry from an identical codebook, thus reconstructing an approximating to the original signal. Compression is obtained because transmitting the address of a codebook entry requires fewer bits than transmitting the vector itself [Huang, Nasrabadi, Midanda-Trigueros, Kumar].

A surprising number of everyday problems are difficult to solve by traditional algorithms. A problem may qualify as difficult for a number of different reasons; for example, the data may be too noisy or irregular; the problem may be difficult to model; or it may simply take too long to solve. It's easy to find examples: finding the shortest path connecting a set of cities, dividing a set of different tasks among a group of people to meet a deadline, or fitting a set of various sized boxes into the fewest trucks. In the past, programmers might have crafted a special-purpose program for each problem; now they can reduce their time significantly by using a *genetic algorithm* (*GA*) [Grant, Louis, Al-Rawi, Ryu].

The genetic algorithm is a highly parallel mathematical algorithm that transforms a set (population) of individual mathematical objects (typically fixed-length character (genes) strings), each with an associated fitness value, into a new population (i.e., the next generation) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (i.e., crossover and mutation operations) [Koza].

The genetic algorithm relies primarily on the creative effects of sexual genetic recombination (crossover) and the exploitative effects of the Darwinian principle of survival and reproduction of the fittest. Mutation is a decidedly secondary operation in genetic algorithms.

The four major steps in preparing to use the conventional genetic algorithm (on fixed-length character strings) to solve a problem involve:

1. determining the representation scheme,
2. determining the fitness measure,

3. determining the parameters and variables for controlling the algorithm, and
4. determining the way of designating the result and the criterion for terminating a run.

The crossover operation for genetic algorithm creates variation in the population by producing new offspring that consist of parts taken from each parent. The crossover operation starts with two parental strings of genes and produces two offspring strings of genes. The two parents are chosen from the population by using different selection methods (random selection method from the same cluster in this work). Also there are many selection method of the genes in the parents that are participate in the crossover operation (also random selection method is used in this work).

The mutation operation introduces random changes of structures in the population. In conventional genetic algorithms operating on strings, the mutation operation can be beneficial by reintroducing diversity in a population that may be tending to converge prematurely. Mutation is asexual and operates on only one parental string of genes. The mutation operation begins by selecting a point (gene) at random within the string of genes. The mutation operation then change whatever is currently at the selected point and inserts a randomly generated gene.

The above explaination of the genetic algorithm and its operations is a breif explaination. For more details about the use of genetic algorithm and how to set the parameters and variables that are used to control the work of the genetic algorithm we advise to return to [Koza].

2. VQ Using Genetic Algorithm (GAVQ): A Proposed Algorithm

In this section, a new algorithm for data compression is suggested. This algorithm tries to exploit the facilities of Genetic Algorithm (GA) and then uses these facilities to make the performance of the Vector Quantization technique more efficient and powerful. The main steps of the proposed algorithm can be stated as follows:

Step1: Problem Representation (focusing on the choice of a suitable representation of the problem).
Step2: Clustering (grouping the most similar input instances in sets that have common characteristics between these input instances).

Step3: Genetic Operations (performing the Crossover and Mutation operations on the elements of the sets that are produced from Step2).
Step4: Merging (merging those sets becomes nearest after performing the genetic operations in Step3).
Step5: Performance Evaluation and Termination Criteria (testing the performance of the algorithm at each generation and termination criteria, if the termination criteria are satisfied then STOP, otherwise GOTO Step3).

2.1 Problem Representation

The first step of the GAVQ algorithm is the preparing step, which involves the representation of the problem in such a way that it becomes suitable to be applied by the GAVQ algorithm.

2.1.1 Vector Representation

Initially, the sound data file is divided into a sequence of vectors of D-dimension (for examples 4, 8, 12, 16, 24, 32 or 64). It is important to mention here that the algorithm look to the sound data file as a file that contains a sequence of unsigned bytes (i.e., a sequence of values between 0 and 255). Now, consider the case that one has to represent a given D-dimensional input vector $\vec{X}_i = (x_0, x_1, x_2, \ldots, x_{D-1})$ with a particular codeword $\vec{C}_j = (c_0, c_1, c_2, \ldots, c_{D-1})$ selected as the best representative of the vector $\vec{X}_i$ within a codebook (i.e., the codevectors are extracted from the input vectors). The selection is based on the minimum distance criterion. The VQ approach is the process that finds a mapping from the set of *N* input vectors $(\vec{X}_0, \vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{N-1})$ to the *M* output vectors (codewords or codevectors) $(\vec{C}_0, \vec{C}_1, \vec{C}_2, \ldots, \vec{C}_{M-1})$, these *M* codevectors represent the codebook as shown in figure (1).

2.1.2 Chromosomal Representation

For any GA, a chromosome representation is needed to describe each individual in the population. The representation scheme determines how the problem is structured in the GA and also determines the genetic operators that are used. Each individual or chromosome is made up of a sequence of genes from a certain alphabet (i.e., letters, integer numbers,

floating point number or bytes). Consider an input vector $\vec{X}_i$ = (x_0, x_1, x_2, …, x_{D-1}) be an individual (chromosome) in the population that consists of *N* individuals (input vectors), each chromosome consists of D number of unsigned bytes (genes), so the chromosome contains a string of (D*8) binary digits or bits (i.e., 0 or 1), where (D*8) represents a chromosome length, as shown in figure (1).

Genetic algorithms use a number of parameters to control their evolutionary search for the solution to their given problem. These parameters include selection operator, crossover operator, mutation operator, crossover rate, mutation rate and maximum number of generations.

At the start of the proposed algorithm, the population size is equal to *N,* which contains all input vectors ($\vec{X}_0, \vec{X}_1, \vec{X}_2, ..., \vec{X}_{N-1}$), and we note that at start of the algorithm the codebook is empty so *M*=0, which will contain the codevectors ($\vec{C}_0, \vec{C}_1, \vec{C}_2, ..., \vec{C}_{M-1}$) at the end of the compresion operation.

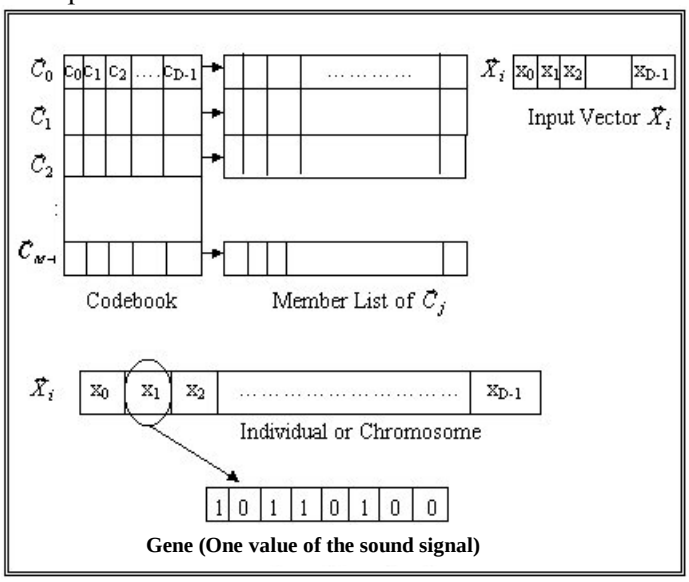

Figure (1): Problem Representation.

2.2 Clustering

This operation assigns each vector $\vec{X}_i$ in the population to the codevector $\vec{C}_j$ (for j = 0,1,2, …, M-1) according to the similarities between $\vec{X}_i$ and $\vec{C}_j$, in another words, a vector $\vec{X}_i$ is assigned to the codevector $\vec{C}_j$ if $\vec{C}_j$ is the nearest codevector to $\vec{X}_i$, such that:

$$D(\vec{X}_i, \vec{C}_j) \le D(\vec{X}_i, \vec{C}_l) \quad (j, l=0,1\ldots M\text{-}1), (i=0,1\ldots N\text{-}1), j \ne l \quad \textbf{...(1)}$$

where $D(\vec{X}_i, \vec{C}_j)$ means the distortion measure between two vectors of D-dimension given as:

$$\mathrm{D}(\vec{\mathrm{X}}_{\mathrm{i}}, \vec{\mathrm{C}}_{\mathrm{j}}) = \frac{1}{\mathrm{D}} \sum_{\mathrm{k}=0}^{\mathrm{D}-1} \left| \vec{\mathrm{x}}_{\mathrm{ik}} - \vec{\mathrm{c}}_{\mathrm{jk}} \right| \quad \textbf{.....(2)}$$

To explain this step, when the algorithm start (i.e., in the first generation). The algorithm calculates the distortion, between the vectors $\vec{X}_i$ (for i=0,1,…,N-1) and the codevectors in the codebook, by using the equation (1) and (2) and check the following cases:

1. If there are no codevectors in the codebook (i.e., when M=0 in the start of the first generation), the algorithm set the vector $\vec{X}_i$ as the first codevector in the codebook. This case is done especially for the vector $\vec{X}_0$ only.
2. If there are some codevector in the codebook, the algorithm try to find the vector $\vec{C}_j$ from the equation (1). After that the algorithm checks:

a) If the $D(\vec{X}_i, \vec{C}_j) \le \varepsilon$, where ε is a small number less than one (for example: if the vector dimension D=100 then ε = 0.05. But in this algorithm when the vector dimension is selected to be D=8 as in table(1), the value of ε = 0.4). Then a vector $\vec{X}_i$ is assigned to the codevector $\vec{C}_j$ because it is similar to this codevector.

b) If the $D(\vec{X}_i, \vec{C}_j) > \varepsilon$. Then a vector $\vec{X}_i$ is add as a new codevector in the codebook and set M=M+1.

The selection of the nearest codevector needs search through all the individuals in the population using a full sequential search. This is done

because the algorithm considered firstly that all the input vectors are codevectors (i.e., *M*=*N*). This may take a long time (for a large number of input vectors) because the clustering operation is done only once at the start of the algorithm.

The clustering operation produces to the next step (i.e., Step 3) a codebook of *M* codevectors ($\vec{C}_0, \vec{C}_1, \vec{C}_2, ..., \vec{C}_{M-1}$), where $M \leq N$. Each codevector $\vec{C}_j$ involves a list of indexes (members) that represent all input vectors (members) which belong to this codevector.

2.3 Genetic Operations

Genetic operations (crossover and mutation) are applied to each element $\vec{C}_j$ of the codebook to get a more suitable codevector that represents its members.

Initially, before the start of the algorithm, the parameters that control the working of the GA must be set. The *random selection* of individuals (parent(s)) for genetic operations is considered to be the selection operator. The crossover operator is a *single point* (*byte or value*) in each individual (parent). The *complement method* is considered to be the mutation operator. The crossover and mutation are applied on the population in rates %50 for each. The maximum number of generations assumed 20.

2.3.1 Crossover Operation

For each codevector $\vec{C}_j$ in the codebook, select randomly from its member's list one index for example *i* (i.e., one input vector $\vec{X}_i \in \{\vec{C}_j\}$) as a first parent and assume that the codevector $\vec{C}_j$ itself is to be the second parent. Then one point (byte), in each of the above selected parent, is randomly selected (x_{ik} and c_{jk} for k = 0,1,2,…,D-1) and then exchange these points between the two parents to produce two child (i.e., two new vectors) $\vec{V}_1$ and $\vec{V}_2$. Note that in this algorithm, any popular algorithm that is used to generate random numbers can be used here.

After that, the checking step begins to check the most adequate vector among the vectors ($\vec{C}_j, \vec{V}_1$ and $\vec{V}_2$), to replace it as a new codevector $\vec{C}_j$ that represents its member's list as a better codevector, and ignore the two others. This check process is performed by calculating the total distortion

over the members (vectors) that belong to the codevector $\vec{C}_j$, for each one of the three vectors ($\vec{C}_j$, $\vec{V}_1$ and $\vec{V}_2$). The Total Distortion (*TD*) can be calculated as:

$$TD(\vec{Y}) = \sum_{i=0}^{E-1}\sum_{j=0}^{D-1}\left|\vec{y}_j - \vec{x}_{ij}\right| \text{.....(3)}$$

Where:
E is the number of members (vectors) in the list of members of the codevector $\vec{C}_j$.

D is the vector dimension (i.e., chromosome length in byte).

$\vec{Y}$ is represent one of the three vectors ($\vec{C}_j$, $\vec{V}_1$, $\vec{V}_2$).

The crossover operation is illustrated in figure (2).

2.3.2 Mutation Operation

For each codevector $\vec{C}_j$ in the codebook, select randomly one point (gene) and complement it. Then, replace the new gene in the place of the old one to produce a new vector $\vec{V}$. Now, calculate the total distortion for the new vector $TD(\vec{V})$ as in equation (3) and compare it with the total distortion $TD(\vec{C}_j)$ for the original codevector. The new vector $\vec{V}$ is set as a new codevector in place of the original codevector $\vec{C}_j$ if $TD(\vec{V}) \le TD(\vec{C}_j)$, otherwise ignore this new vector $\vec{V}$. The mutation operation is illustrated in figure (3).

For more details, in this algorithm each vector is consider to be consist of a set of genes (i.e., unsigned bytes (0..255)). This assumption allows the algrithm to treat the sound values, in the sound signal, without regarding to their orginal vlaues (if they are positive or negative). For example: if the source sound value, in the sound signal, is (–30) and it is obviously represented as a signed byte. But the algorithm consider that this sound value is represented as unsigned value (226). Therefore, there is no problem if the algorithm use the complement operation $|\mathbf{255\text{-}c_3}|$ in figure (3). For examples: if c_3=0 then |255-0|=255, and if c_3=255 then |255-255|=0. This means that any new value of $\mathbf{c_3}$, in the new vector $\vec{V}$, that is produces from this complement operation between (0…255) and will

cause to accept or reject the vector $\vec{V}$ after checking the relation $TD(\vec{V}) \leq TD(\vec{C}_j)$.

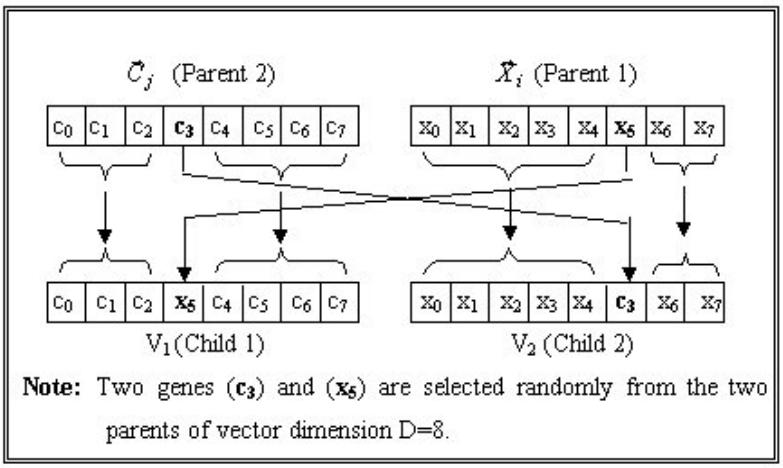

Figure (2): Crossover Operation.

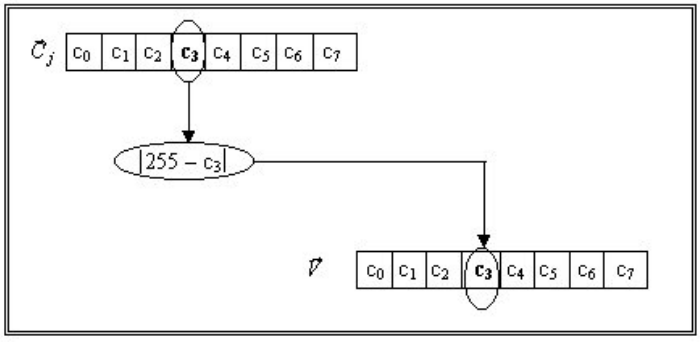

Figure (3): Mutation Operation.

2.4 Merging

The goal of this step is to merge each two codevectors $\vec{C}_i$ and $\vec{C}_j$ if the rate of the distortion per each two genes within the two codevectors

$(D(\vec{C}_i, \vec{C}_j)) \leq \varepsilon$, where ε is a small number less than one (0.4 in this algorithm), D is the vector dimension and $i, j = 0,1,2, \ldots, M\text{-}1$ (where M is the number of codevectors in the codebook). We must note here that from generation to generation, the merging operation try to minimize the number of codevectors in the coodebook. But it is necessary to know that the number of codevectors that will passed to the next generation may be from 1 to any number less or equal M.

2.5 Performance Evaluation and Termination Criteria

After each generation, some performance evaluation and termination measures are calculated to decide if the algorithm goes to do a next generation or to stop. Signal to Noise Ratio (*SNR*), Peak Signal to Noise Ratio (*PSNR*) should be calculated between each input vector and the reproduction codevector within the codebook that these input vectors belong to. This means that these measures are calculated between each byte in the source sound file and the reconstructed sound file after the de-compression operation is done. For simplicity, let $b(i)$ and $\hat{b}(i)$ be the byte in the sequence i within the source and decompression sound data file respectively, and S be the size in byte for the source and decompression sound data file.

$$SNR_{db} = 10\log_{10}\left[\frac{\sum_{i=0}^{S-1} b^2(i)}{\sum_{i=0}^{S-1}\left[\hat{b}(i) - b(i)\right]^2}\right] \quad \text{.....(4)}$$

$$PSNR_{db} = 10\log_{10}\left[\frac{\left[\text{peak value of } b(\text{i})\right]^2}{\sum_{i=0}^{S-1}\left[\hat{b}(i) - b(i)\right]^2 \Big/ S}\right] \quad \text{.....(5)}$$

The peak value of $b(i)$ is the total dynamic range of the input sound data.

To check the compression performance, the values of Compression Ratio (*CR*) is calculated. The compression ratio is the amount of compression and is calculated in two forms, On-line (when the codebook is taken into

consideration) and Off-line (when the codebook is not taken into consideration). In this algorithm the on-line CR is used:

- On-line:

$$CR = \frac{\text{source sound file size}}{\text{VQ file size} + \text{codebook size}} \quad \text{……..…..(6)}$$

- Off-line:

$$CR = \frac{\text{source sound file size}}{\text{VQ file size}} \quad \text{…...…….…….(7)}$$

The better compression performance of the algorithm is with the highest compression ratio *CR* and the highest *SNR* that can the algorithm be satisfy, obviously this is mainly depending on the nature of the sound data.

The termination criterion in this algorithm is that the algorithm will stop on at least if one of the following condition is satisfied:

1. SNR ≥ 22.0 and CR ≥ 4.0 or.
2. The number of generation reached the Max, (where Max is the maximum number of generations that are allowed in the algorithm (i.e., to prevent the compression operation take a very long time)).

3. Experimental Results

The performance of using GA in VQ technique for data compression has been tested by applying it to a set of various sound and speech data files. The performance of the proposed algorithm is evaluated in terms of the number of codewords required to achieve a particular *SNR* while encoding a number of test data files. The following table show the results that are obtained after applying the (GAVQ) proposed algorithm and the one of the classical VQ algorithms (k-means in this table) to the some (sound and speech) data files. It is necessary to clearify that both GAVQ and k-means algorithm are considered firstly that all the input vectors are codevectors and each of them try to find as little as possible number of codevectors, that can be used to represent all the input vectors of the sound data. To satisfy that, both algorithms depend on the same termination criterias that are mentioned above. All programs were written by using Visual C++ language (version 6.0) and these programs were executed on the Pentium III (500 MHz) personal computer.

Table (1): Comparison Table between GAVQ and Classical k-means VQ Compression Algorithm.

File	Compression Algorithm	SNR (db)	PSNR (db)	On-line CR (x:1)	Vector Dimension 8 byte		No. of Crossover Operations	No. of Mutation Operations	No. of Generations	Time of Compression Operation (Second)
					No. of Code Vectors	No. of Input Vectors				
Music Signal	GAVQ	22.77	28.44	6.85	89	2422	19	275	5	0.5
	VQ (k-means)	23.33	29.04	5.33	152	2422	-	-	-	0.20
Music Signal	GAVQ	22.74	27.10	10.40	34	14392	9	109	8	2.8
	VQ (k-means)	22.74	27.15	10.31	47	14392	-	-	-	9.63
Arabic Speech Signal	GAVQ	22.37	28.48	9.59	64	6090	35	144	4	0.71
	VQ (k-means)	23.40	29.53	8.17	79	6090	-	-	-	1.83
Arabic Speech Signal	GAVQ	22.14	25.13	7.63	125	5759	130	895	6	1.88
	VQ (k-means)	23.27	26.28	6.15	216	5759	-	-	-	0.88
English Speech Signal	GAVQ	22.56	28.54	6.51	217	7579	100	604	10	3.02
	VQ (k-means)	23.50	29.51	5.61	284	7579	-	-	-	2.02

From the above comparison table, we can note :

1. The GAVQ algorithm success to increase CR with saving the SNR in the acceptable range.
2. The number of codevector is less than the classical k-means algorithm.
3. The number of crossover and the mutation operations recorded in the above table represent the successful crossover and mutation operations that are participate in increase the CR. Because the algorithm excludes

any crossover and mutation operations that are not cause to increase the CR and saving the SNR at the acceptable value. In other words, the algorithm is excluded any new vector produced from crossover and mutation operation when its total distortion (TD) not satisfying the required conditions as this is mentioned in (section 2.3.1 and section 2.3.2).

4. Inspite of the major problem (i.e., the long time) of using GA in any application. The required time for compression when using the GA (in this work), by mixing it with the VQ method, is near to the required time by the k-means. Certainly, it is not suitable to use this algorithm in the real-time systems because the genetic algorithm required long time (especially when the population become very large).

5. The above table presents some examples of sound and speech file of small size. Certainly, when using a file of large size, the algorithm firstly divide the file into a number of segments and then apply the steps of the algorithm on these segments to reduce the amount of time and memory space that are required to complete the compression operation.

6. Obviously, when we used the SNR and PSNR measures to calculate the distortion (noise) that will appear in the sound signal during the compression operation is not enough to ensure that this noise not causes a bad effects on the hearing (and then understanding) the sound or speech. We can ensure the effect of this noise by adding another subjective test to the algorithm, but for comparing the performance of the proposed algorithm and another classical algorithm, it is obvious in the first step of the comarision to use objective test as SNR and PSNR.

4.Conclusion

Vector Quantization (VQ) compression method was first studied and implemented, then the GAVQ algorithm that makes use of the GA to design the VQ codebook was proposed, it exploits the crossover and the mutation operations of the genetic algorithm. The GAVQ algorithm was tested on some sound and speech data files. The recorded results showed that the idea of mixing between the genetic algorithm and the vector quantization compression method enhances the performance of the (VQ) method alone.

The conclusions that can be drawn from this work is that when mixing the (GA) with Vector Quantization method (VQ), we can get a good

enhancement for the performance of this method in term of increasing the compression ratio and saving the *SNR* in the acceptable level.

We must mentioned here that the proposed (GAVQ) algorithm was previously applied on the image files and its prove that is good to use it in compressing the image files. In the future, after these successes of application of this algorithm. We will try to apply this algorithm on the multimedia files (or video files) which are contain a mixed data of sound and image files

References

- Al-Rawi Hisham, Jane J. Stephan, "*Genetic Algorithm Based Image Segmentation*", Proceeding of CATAEE'99, Philadelphia University, Jordan, 1999.
- Cabral Jim, *"3D Vector Quantization of Magnetic Resonance Images"*, Internet Paper, http://www.data-compression/vq.html, 1994.
- Chan Yuk-Hee, Wan-Chi Siu and Kin-Man Lam, *" A Novel VQ Encoding Algorithm Based on Adaptive Searching Sequence"*, IEEE International Symposium on Speech, Image Processing and Neural Networks, 13-16 April 1994, Hong Kong.
- Grant Keith, *"An Introduction to Genetic Algorithms"*, C/C++ Users Journal, pp. 45-58, March, 1995.
- Huang C. M., Q. Bi, G. S. Stiles, *"Fast Full Search Equivalent Encoding Algorithms for Image Compression Using Vector Quantization"*, IEEE Transaction on Image Processing, Vol. 1, No. 3, Julay 1992.
- Koza John R., *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"*, 1992.
- Kumar Rajeev, *"Codebook Design for Vector Quantization Using Multiobjective Genetic Algorithms"*, Internet Paper, http://www.rdg.ac.uk/~ssr97jdk/MPSN/Kumar1codebookMPSN.ps.gz, 1999.
- Liang K. M., C. M. Huang, and R. W. Harris, *"Compression Between Adaptive Search and Bit Allocation Algorithms for Image compression Using Vector Quantizetion"*, IEEE Transaction on Image Processing, Vol. 4, No. 7, July 1995.
- Louis Sushil J., *"Genetic Algorithm and Design"*, Internet Paper, http://www.cs.unr.edu/~sushil/papers/thesis/thesishtml/node2.html, 1997.

- Midanda-Trigueros Armando, Jesus-M. Val-Bueno, and Anibal-R. Figueiras-Vidal, *"The multiple Representation Problem in Genetic Approaches for Index Assignment in Vector Quantization Codebook Design"*, Internet Paper, http://www.ehis.nary.mil/tp/humanscience/papers/art18.pdf, 1999.
- Nasrabadi Nasser M., Robert A. King, *"Image Coding Using Vector Quantization: A Review"*, IEEE Transaction on Communication, Vol. 36, No. 8, Augest 1988.
- Ryu Tae-Wan, Christoph F. Eick, *"MASSON: Discovering Commonalties in Collection of Objects Using Genetic Programming"*, Internet Paper, URL: http://www.cs.uh.edu/~twryu.
- Xiaolin Wu and Jiang Wen, *"Conditional entropy coding of VQ indexes for image compression"*, IEEE Transaction on Image Processing, vol. 8, no. 8, pp. 1005-1013, Augest 1999.

Received,12 May, 2005.