

Image Compression Using Vector Quantization and Genetic Algorithms

Salah Awad Salman

Department of Computer Science, College of computers, Al-Anbar University

Email: salah_eng1996@yahoo.com

ABSTRACT

Image compression involves reducing the size of image data file, while retaining necessary information. This paper uses the facilities of the Genetic Algorithm for the enhancement of the performance of one of the popular compression method, Vector Quantization method is selected in this work. After studying this method, new proposed algorithm for mixing the Genetic Algorithm with this method was constructed and then the required programs for testing this algorithm was written. The proposed algorithm was tested by applying it on some image data files. Some fidelity measures are calculated to evaluate the performance of the new proposed algorithm. A good enhancement was recorded for the performance of the Vector Quantization method when mixed with the Genetic Algorithm. All programs were written by using Matlab (version 7.0) and these programs were executed on the Pentium III (866 MHz) personal computer.

Keywords:

Image Compression, Vector Quantization, Genetic Algorithm

ضغط الصور باستخدام التقريب المتجهي والخوارزميات الجينية

صلاح عواد سلمان

قسم علوم الحاسبات، كلية الحاسبات، جامعة الانبار

الخلاصة

ضغط الصورة يتضمن تقليل حجمها مع الحفاظ على العناصر الضرورية فيها . هذا البحث يستخدم تسهيلات الخوارزميات الجينية لتحسين أداء إحدى طرق الضغط الشائعة. طريقة (VQ) تم اختيارها في هذا العمل . بعد دراسة هذه الطريقة تم اقتراح خوارزمية جديدة تجمع بينها وبين الخوارزميات الجينية وتم بناء كافة البرامجيات الضرورية لأختبار هذه الخوارزمية المقترحة على عدة صور . تم تسجيل تحسن جيد بأداء هذه الخوارزمية المقترحة . بعض المقاييس تم حسابها لتجيك أداء الخوارزمية المقترحة . كل البرامجيات كتبت باستخدام (Matlab ver.7) وهذه البرامجيات نفذت على حاسبة بنتيوم III (866 MHz) .

1. INTRODUCTION

Image compression has been pushed to the forefront of the image processing field. This is largely a result of the growth in computer power, the corresponding growth in the multimedia market, and the advent of the world wide web, which makes the internet easily accessible for every one. Additionally, the advances in video technology, including high-definition television, are creating a demand for new, better, and faster image compression methods[1].

Vector quantization is one of these methods, that 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 [2]. Vector Quantization has been successfully used in speech and image data compression [3]

This paper is organized as follows. In sections 2 and 3, an introduction is given to the main ideas of Vector Quantization and genetic algorithms respectively. In section 4, the proposed method is described. Section 5 will be dedicated to the application of the proposed method to images. Experimental results are discussed. Finally , conclusion and perspectives of the work are suggested.

2. VECTOR QUANTIZATION

One of the emerging technologies for lossy data (image in this work) compression is Vector Quantization. Vector Quantization can be used to take advantage of the correlation between neighboring pixels by quantizing pixels 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 distortion measure is simply the mean squared error between the quantized pixels and the codevector [4].

Vector quantization is the process of mapping which can have many values to vector that has a smaller (quantized) number of values. For image compression, the vector corresponds to a small subimage, or block. The previous types of quantization deal with taking a single value and reducing the number of bits used to represent that value, this is called scalar quantization and is most easily achieved by rounding or truncation. Vector quantization treats the entire subimage (vector) as a single entity and quantizes it by reducing the total number of bits required to represent the subimage. This is done by utilizing a codebook, which stores a fixed set of vectors, and the coding the subimage by using the index (address) into the codebook [1] 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 [5,6,7,8].

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 [9,10,11,12,13].

3. GENETIC ALGORITHMS

Genetic algorithms are general-purpose search algorithms that use principles inspired by natural population genetics to evolve solutions to problems. They were first proposed by Holland [14]. A GA operates on a population of randomly generated solutions, chromosomes often represented by binary strings. The population advances toward better solutions by applying genetic operators, such as crossover and mutation. In each generation, favorable solutions generate offspring that replace the inferior individuals. Crossover hybridises the genes of two parent chromosomes in order to exploit the search space and constitutes the main genetic operator in GAs. The purpose of mutation is to maintain the diversity of the gene pool. An evaluation or fitness function plays the role of the environment to distinguish between good and bad solutions [15].

3.1 Crossover

Exchanges corresponding genetic material from two parents, allowing useful genes on different parents to be combined in their offspring. Two parents may or not be replaced in the original population for the next generation in different strategies. Crossover is the key to genetic algorithm's power. Most successful parents reproduce more often.

3.2 Mutation

Random mutation provides background variation and occasionally introduces beneficial material into species chromosomes. Without the mutation, all the individuals in population will eventually be the same (because of exchange of genetic material) and there will be no progress anymore. We will be stuck in local maximum as it is used to say. Mutation in case of binary string is just inverting a bit .

3.3 Elitism Technique

In this technique the fit individual in $g(t)$ is copied directly to $g(t+1)$ without being changed by the other operations. This operator is sometimes used to try to make sure that there will be a reasonable fit individual present in the population at every time step. It helps to avoid having all the string get modified by crossover and mutation in away that no good solution exists at some time (t) , such individuals can be lost if they are not selected to reproduce or if they are destroyed by crossover or mutation [3,11].

Many researchers have found that elitism significantly improves the GA's performance. The operation is easily performed, the selected individuals are simply copied and inserted into the new population, the number of selected individuals should be limited [30].

3.4 Genetic Algorithm Parameters

In order to converge, several parameters of genetic algorithm have to be considered:

3.4.1 Crossover and Mutation Rate

In most cases, only a few of the component bits are mutated since the mutation rate is set at a very low value. This ensures that the mutation operator plays only a background role in the genetic algorithms as opposed to the crossover operator.

3.4.2 Population Size

For algorithms convergence, bigger population is better, because we have more genetic material for selection and reproduction. Changes to build better individuals are bigger.

However, there is one important aspect to consider though aspect of algorithms speed. Bigger population means more evaluation. Every individual in population has to be measured in terms of fitness. Usually this is computationally most expensive procedure in genetic algorithm. Therefore, population size is kept about 32. Of course it depends on evaluation complexity.

3.4.3. Number of Genetic Generations

It depends on tasks complexity, sometimes hundred generations is enough, another time thousands of generations isn't enough. It is probably wise to stop when no improvement in solution quality is made for certain time. Limit condition can be calculated taking into account tasks dimensions. For complex task, we may allow more generations.

4. THE PROPOSED ALGORITHM (VECTOR QUANTIZATION AND GENETIC ALGORITHM):

In this section, a new algorithm for Image 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 Elitism Technique.
- Step6:** Termination Criteria (testing the performance of the algorithm at each generation and termination criteria, if the termination criteria are satisfied then STOP, otherwise GOTO Step2).

4.1 Problem Representation

The first step of the proposed 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 algorithm.

4.1.1 Vector Representation

Initially, the image is divided into a non-overlapped blocks of dimension $H \times W$ (for examples 2×2 , 2×4 , 2×6 , 4×4 , 4×6 , 4×8 or 8×8). Each block is then represented as a vector of D -dimension (i.e., $D = H \times W$). Now, consider the case that one has to represent a given D -dimensional input vector $\bar{x}_i = (x_1, x_2, x_3, \dots, x_D)$ with a particular codeword $\bar{c}_j = (c_1, c_2, c_3, \dots, c_D)$ selected as the best representative of the vector \bar{x}_i within a codebook (i.e., the codevectors are extracted from the input vectors). The selection is based on the minimum Euclidean distance criterion. The Vector Quantization approach is the process that finds a mapping from the set of N input vectors $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \dots, \bar{x}_N)$ to the M output vectors (codewords

or codevectors) $(\bar{c}_1, \bar{c}_2, \bar{c}_3, \dots, \bar{c}_M)$, these M codevectors represent the codebook as shown in figure (1).

4.1.2 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 $\bar{x}_i = (x_1, x_2, x_3, \dots, x_D)$ be an individual (chromosome) in the population that consists of N individuals (input vectors), each chromosome consists of D number of 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 $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \dots, \bar{x}_N)$. Also, the codebook itself contains all input vectors as the codevectors $(\bar{c}_1, \bar{c}_2, \bar{c}_3, \dots, \bar{c}_M)$ initially (i.e., $M=N$).

4.2 Clustering

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

$$D(\bar{X}_i, \bar{C}_j) \leq D(\bar{X}_i, \bar{C}_l) \quad (1)$$

Where : $(j, l=0, 1 \dots M-1), (i=0, 1 \dots N-1), j \neq l$

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

$$D(\bar{X}_i, \bar{C}_j) = \sum_{k=0}^{D-1} |x_k - c_k| \quad (2)$$

The selection of the nearest codevector needs search through all the individuals in the population using a full sequential search. This may take a long time (for a large number of input vectors) at the first generation, but it becomes suitable more and more after some generations.

The clustering operation produces to the next step (i.e., Step 3) a codebook of M codevectors $(\bar{c}_1, \bar{c}_2, \bar{c}_3, \dots, \bar{c}_M)$, where $M \leq N$. Each codevector \bar{c}_j involves a list of indexes (members) that represent all input vectors (members) which belong to this codevector.

4.3 Genetic Operations

Genetic operations (crossover and mutation) are applied to each element 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) in each individual (parent) and the Elitism Technique also used to improve the GA's performance. The complement method is considered to be the mutation operator. The crossover and mutation are applied on the population in rates 0.7 and 0.2 respectively. The maximum number of generations is considered 15.

4.3.1 Crossover Operation

For each codevector \bar{c}_j in the codebook, select randomly from its member's list one index for example i (i.e., one input vector \bar{x}_i) as a first parent and assume that the codevector \bar{c}_j itself is to be the second parent. In the next step, one point (byte) in each parent is randomly selected (x_k and c_k for $k = 0, 1, 2, \dots, D-1$) and then exchange these points between the two parents to produce two child (i.e., two new vectors) \bar{v}_1 and \bar{v}_2 .

After that, the checking step begins to check the most adequate vector among the vectors (\bar{c}_j, \bar{v}_1 and \bar{v}_2), to replace it as a new codevector \bar{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 \bar{c}_j , for each one of the three vectors (\bar{c}_j, \bar{v}_1 and \bar{v}_2). The Total Distortion (TD) can be calculated as:

$$TD(\bar{Y}) = \sum_{i=0}^{E-1} \sum_{j=0}^{D-1} |y_{ij} - x_{ij}| \quad (3)$$

where E is the number of members (vectors) in the list of members of the codevector \bar{c}_j , and D is the vector dimension (i.e., chromosome length in byte). The crossover operation is illustrated in figure (2).

4.3.2 Mutation Operation

For each codevector \bar{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 \bar{v} . Now, calculate the total distortion for the new vector $TD(\bar{v})$ as in eq. (3) and compare it with the total distortion $TD(\bar{c}_j)$ for the original codevector. The new vector \bar{v} is set as a new codevector in place of the original codevector \bar{c}_j if $TD(\bar{v}) \leq TD(\bar{c}_j)$, otherwise ignore this new vector \bar{v} . The mutation operation is illustrated in figure (3).

4.4 Merging

The goal of this step is to merge each two codevectors \bar{c}_i and \bar{c}_j if the rate of the distortion per each two genes within the two codevectors $(D(\bar{c}_i, \bar{c}_j) / D) \leq \epsilon$, where ϵ is a small number less than one (0.05 in this algorithm), D is the vector dimension and $i, j = 0, 1, 2, \dots, M-1$ (where M is the number of codevectors in the codebook).

4.5 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*) and Normalized Mean Squared Error (*NMSE*) should be calculated between each input vector and the reproduction codevector within the codebook that these input vectors belong to. This means really that these measures are calculated between each pixel in the source image file and the reconstructed image file after the de-compression operation is done. For simplicity, let $b(i, j)$ and $\hat{b}(i, j)$ be the pixel in the raw i and column j within the source and decompression image file respectively.

$$(4) \text{SNR}_{\text{db}} = 10 \log_{10} \left[\frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \hat{b}^2(i, j)}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [\hat{b}(i, j) - b(i, j)]^2} \right]$$

$$\text{PSNR}_{\text{db}} = 10 \log_{10} \left[\frac{[\text{peak value of } b(i, j)]^2}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [\hat{b}(i, j) - b(i, j)]^2} \times N^2 \right] \quad (5)$$

$$\text{NMSE} = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [b(i, j) - \hat{b}(i, j)]^2}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [b^2(i, j)]} \times 100 \quad (6)$$

The peak value of $b(i, j)$ is the total dynamic range of the input image and N is the source image dimension.

To check the compression performance, the values of Compression Ratio (*CR*) and Bit Per Pixel Rate (*BR*) are calculated. The compression ratio is the amount of compression, while the BR rate is the number of bits required to represent each pixel value of the compressed image. They are calculated in two forms, On-line (when the codebook is taken into consideration) and Off-line (when the codebook is not taken into consideration):

•On-line:

$$\text{BR} = \frac{\text{VQ file Size in bits} + \text{codebook size in bits}}{\text{source image file size in pixel}} \text{ bits/pixel} \quad (7)$$

$$CR = \frac{\text{source image file size}}{\text{VQ file size} + \text{codebook size}} \quad (8)$$

•Off-line:

$$BR = \frac{\text{VQ file size in bits}}{\text{source image file size in pixel}} \quad \text{bits/pixel} \quad (9)$$

$$CR = \frac{\text{source image file size}}{\text{VQ file size}} \quad (10)$$

The better compression performance of the algorithm is with the highest compression ratio (the least *BR*) and the highest *PSNR*.

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

1. $SNR \geq 22.0$ and $CR \geq 4.0$ or.
2. The number of generation reached the Max, (where Max =15).

5. EXPERIMENTAL RESULTS

The performance of using GA in Vector Quantization technique for image compression has been tested by applying it to a set of 256 gray-scale levels digital images of size 256×256 pixels. 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 tables show the results that are obtained after applying the proposed algorithm to a set of some image data files.

In the following tables (1)-(4), when the proposed algorithm starts to compress these image files, we note from generation to generation that the compression ratio (CR) is increased with some decreasing in the *SNR*. And after some generations, when the first condition of the termination criteria is satisfied, the algorithm will stopped. If the first condition was not satisfied then the algorithm will continue until the second condition of the termination criteria satisfy. The number of generations that are needed for completing the compression phase is depend on the nature of the data file.

6.CONCLUSION

Vector Quantization compression method was first studied and implemented, then the new method that makes use of the GA to design the Vector Quantization codebook was proposed, it exploits the crossover, elitism technique and the mutation operations of the genetic algorithm. The proposed algorithm was tested on some image 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 Vector Quantization method alone.

The conclusions that can be drawn from this work is that when mixing the (GA) with Vector Quantization method, we can get a good enhancement for the performance of this method in terms of convergence time and CR.

REFERENCES

- [1] S. E. Umbaugh, "Computer Vision and Image Processing", Prentice Hall, 1998.

- [2] Yuk-Hee Chan, Wan-Chi Siu and Kin-Man Lam, “ A Novel Vector Quantization Encoding Algorithm Based on Adaptive Searching Sequence”, IEEE International Symposium on Speech, Image Processing and Neural Networks, April 1994,
- [3] K. M. Liang, C. M. Huang, and R. W. Harris, “Compression Between Adaptive Search and Bit Allocation Algorithms for Image compression Using Vector Quantization”, IEEE Transaction on Image Processing, July 1995.

- [4] Jim Cabral, “3D Vector Quantization of Magnetic Resonance Images”, Internet Paper, <http://www.data-compression/vq.html> , 1994.
- [5] C. M. Huang, Q. Bi, G. S. Stiles, “Fast Full Search Equivalent Encoding Algorithms for Image Compression Using Vector Quantization”, IEEE Transaction on Image Processing, July 1992.
- [6] Nasser M. Nasrabadi, Robert A. King, “Image Coding Using Vector Quantization: A Review”, IEEE Transaction on Communication, August 1988.
- [7] Armando Midanda-Trigueros, 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.
- [8] Rajeev Kumar, “Codebook Design for Vector Quantization Using Multiobjective Genetic Algorithms”, Internet Paper, <http://www.rdg.ac.uk/~ssr97jdk/MPSN/Kumar1codebookMPSN.ps.gz>, 1999.
- [9] Keith Grant, “An Introduction to Genetic Algorithms”, C/C++ Users Journal, pp. 45-58, March, 1995.
- [10] Sushil J. Louis, “Genetic Algorithm and Design”, Internet Paper, <http://www.cs.unr.edu/~sushil/papers/thesis/thesishtml/node2.html>, 1997.
- [11] Hisham Al-Rawi, Jane J. Stephan, “Genetic Algorithm Based Image Segmentation”, Proceeding of CATAEE’99, Philadelphia University, Jordan, 1999.
- [12] Tae-Wan Ryu, Christoph F. Eick, “MASSON: Discovering Commonalties in Collection of Objects Using Genetic Programming”, Internet Paper, URL: <http://www.cs.uh.edu/~twryu>.
- [13] “What is Genetic Programming”, Internet Paper, <http://www.genetic-programming.org>, 1999.
- [14] Holland, J.H., "Adaptation in Natural and Artificial Systems", Ann Arbor, University of Michigan Press, 1975.
- [15] Cordon, O., Herrera, F., Hoffmann, F, and Magdalena, L., "Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases", World Scientific, 2001.

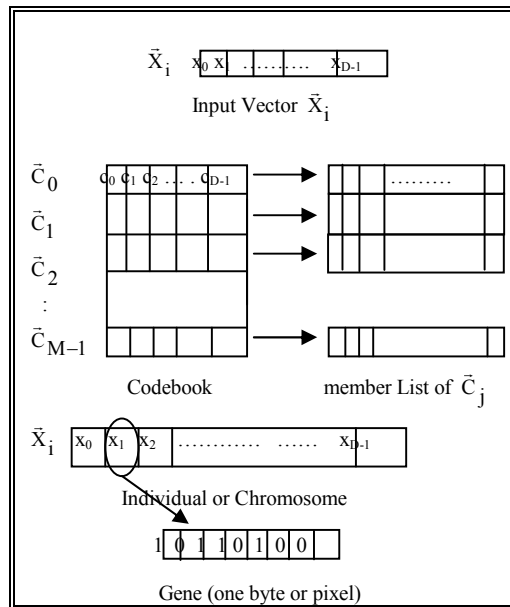


Fig. 1. Problem Representation.

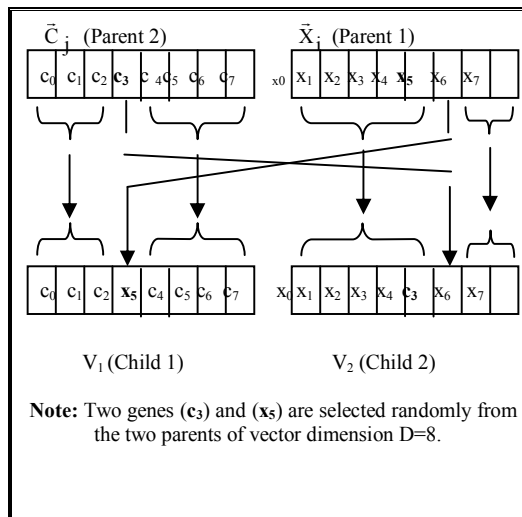


Fig. 2. Crossover Operation.

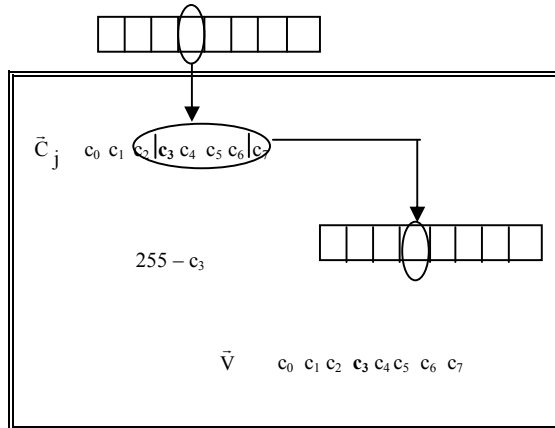


Fig. 3. Mutation Operation.

Table (1): Applying Proposed Algorithm to “Venica” of Vector Dimension 8.

SNR (db)	24.59	23.72	23.41
PSNR (db)	31.11	30.44	30.22
NMSE	1.4507e-5	1.5222e-5	1.6400e-5
CR (x:1)	2.81	3.95	4.21
BR (bpp)	2.82	2.04	1.93
Vector Dimension D = (H×W)	8 (2×4)	8 (2×4)	8 (2×4)
No. of Codevectors in Codebook	1522	860	730
No. of Input Vectors	8192	8192	8192
No. of Generations	2	4	5
Time in Second	390	550	690
Venica.jpg (256×256)			
Original Image		Decompressed Image	

Table (2): Applying Proposed Algorithm to “Venica” of Vector Dimension 16.



SNR (db)	23.71	23.40	23.21
PSNR (db)	30.33	30.19	29.80
NMSE	1.6851e-5	1.9221e-5	2.0111e-5
CR (x:1)	2.21	3.71	4.10
BR (bpp)	2.71	2.10	2.12
Vector Dimension D = (H×W)	16 (4×4)	16 (4×4)	16 (4×4)
No. of Codevectors in Codebook	1060	720	710
No. of Input Vectors	4096	4096	4096
No. of Generations	2	4	5
Time in Second	233.11	488.55	566.21
<p>Venica.jpg (256×256)</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>Original Image</p> </div> <div style="text-align: center;">  <p>Decompressed Image</p> </div> </div>			

Table (3): Applying Proposed Algorithm to “Pas” of Vector Dimension 8.

SNR (db)	23.31	22.71	22.3763
PSNR (db)	31.85	31.42	30.91
NMSE	2.1448e-5	2.8907e-5	3.4116e-5
CR (x:1)	3.21	4.10	5.11
BR (bpp)	2.22	1.6	1.44
Vector Dimension D = (H×W)	8 (2×4)	8 (2×4)	8 (2×4)
No. of Codevectors in Codebook	1155	657	442
No. of Input Vectors	8192	8192	8192
No. of Generations	2	3	4
Time in Second	424	523	612

Pas.jpg (256×256)



Original Image




Decompressed Image


Table (4): Applying Proposed Algorithm to “Pas” of Vector Dimension 16.

SNR (db)	21.66	21.58	22.21
PSNR (db)	30.89	30.99	30.82
NMSE	3.663e-5	4.0145e-5	4.2801e-5
CR (x:1)	2.33	5.12	6.00
BR (bpp)	2.09	1.45	1.30
Vector Dimension D = (H×W)	16 (4×4)	16 (4×4)	16 (4×4)
No. of Codevectors in Codebook	1217	519	390
No. of Input Vectors	4096	4096	4096
No. of Generations	1	3	4
Time in Second	212.5	310.0	413.01

Pas.jpg (256×256)



Original Image



Decompressed Image