Journal of Marine Science and Technology

Volume 18 | Issue 4 Article 10

VECTOR QUANTIZATION BASED ON STEADY-STATE MEMETIC ALGORITHM

Chien-Min Ou
Department of Electronics Engineering, Ching-Yun University, Chungli, Taiwan, R.O.C, cmou@cyu.edu.tw

VECTOR QUANTIZATION BASED ON STEADY-STATE MEMETIC ALGORITHM

Chien-Min Ou*

Key words: memetic algorithm, vector quantizer, steady-state genetic algorithm.

ABSTRACT

A novel memetic algorithm (MA) for the design of vector quantizers (VQs) is presented in this paper. The algorithm uses steady-state genetic algorithm (GA) for the global search and K-means algorithm for the local improvement. As compared with the usual MA using the generational GA for global search, the proposed MA dramatically reduces the computational time for VQ training. In addition, it attains a near global optimal solution, and its performance is insensitive to the selection of initial codewords. Numerical results show that it can save more than 70% of computation time while maintaining a comparable performance as previous MA.

I. INTRODUCTION

Genetic algorithm (GA) [3, 7, 12] is a general-purpose search algorithm for solving optimization problems by simulating natural evolution over populations of candidate solutions. Inspired by biological evolution, the GA consists of a set of genetic strings, which are evaluated by a fitness function. The fittest strings are then regenerated at the expense of the others. Furthermore, crossover and mutation are employed to obtained better strings. The mutation operator changes individual elements of a string, and the crossover operation interchanges parts between strings. In the generational GA, the combination of these operations is called a generation. The evolution of genetic strings may be continued for several generations to obtain a near global optimal solution.

Although the generational GA is effective, it may not be suited for fine tuning search results which are close to optimal. To eliminate this drawback, the memetic algorithm (MA) [1] combining the generational GA with a local search has been proposed. In the MA, the generational GA is used for coarse search, while the subsequent local improvement is then used to refine the generational GA. Its superior performance over pure GA has been found for various applications, such as VLSI design [2], traveling salesman problem [9], and binary quadratic programming [8].

Vector quantization (VQ) [4] has been found to be effective for data compression, pattern recognition and data mining. The VQ techniques remove redundancy in the source, and retain useful information for subsequent processing. A common approach for VQ design is based on K-means algorithm [6], where the codewords and the partition of training set are iteratively optimized one at a time. The K-means algorithm therefore can be viewed as a local optimization approach for VQ design. The basic MA technique using generational GA for global search and K-means for local improvement can be adopted for enhancing the performance of the VQ [11]. However, the computational complexity for the basic MA may be very high. Due to a new set of child strings going to replace the entire set of parent strings in each generation for the subsequent genetic operations in the generational GA.

The objective of this paper is to present a novel MA for VQ design. It employees a steady-state GA [5, 10], instead of the general GA, and results in a low computational complexity. The steady-state GA generates only one child string at a time. A child string with better fitness value will replace its parent string.

The proposed MA is termed the steady-state MA in this paper. To fine tune the global search result of the steady-state GA, the K-means algorithm will be applied for the local improvement of the child string after the crossover and mutation operations. A survival competition of the resulting child string against the existing parent strings then follows. Since only one new genetic string is produced at a time, it is only necessary to employ the K-means algorithm to the new string for local refinement in the steady-state MA. By contrast, in the basic MA, the K-means algorithm should operate over the entire population of genetic strings. The proposed algorithm therefore may have lower computational time for VQ design as compared with the basic MA. Based on the same population size, numerical results reveal that the steady-state MA can save more than 70% of computational time of the basic MA while maintaining a comparable performance of distortion rate for VQ design. The proposed algorithm therefore is an effective alternative for VQ applications where both high performance and low computational time are desired.

Paper submitted 11/07/08; revised 05/08/09; 08/04/09; accepted 08/17/09.
Author for correspondence: Chien-Min Ou (e-mail: cmou@cyu.edu.tw).
**Department of Electronics Engineering, Ching-Yun University, Chungli, Taiwan, R.O.C.*

II. THE PROPOSED ALGORITHM

We first start with the K-means algorithm for the VQ design. Consider a full-search VQ with N codewords $\{\mathbf{y}_1,\ldots, \mathbf{y}_N\}$. Given a set of training vectors $T = \{\mathbf{x}_1,\ldots, \mathbf{x}_t\}$, the average distortion of the VQ is given by

$$D = \frac{1}{wt}\sum_{j=1}^{t} d\left(\mathbf{x}_j, \mathbf{y}_{\alpha(\mathbf{x}_j)}\right). \tag{1}$$

Where w is the vector dimension, t is the number of training vectors, $\alpha(\ )$ is the source encoder, and $d(\mathbf{u}, \mathbf{v})$ is the squared distance between vectors $\mathbf{u}$ and $\mathbf{v}$.

The K-means algorithm is an iterative approach finding the solution of $\{\mathbf{y}_1,\ldots, \mathbf{y}_N\}$ locally minimizing the average distortion D given in (1). The algorithm is based on two necessary conditions for the optimal VQ.

Necessary Condition 1: Let cell T_i be the set of training vectors which are assigned to codeword $\mathbf{y}_i$ by the source encoder $\alpha(\ )$. Given codewords $\{\mathbf{y}_1,\ldots, \mathbf{y}_N\}$, the optimal partition $T_1, T_2,\ldots, T_N$ minimizing average distortion D in (1) should satisfy

$$T_i = \left\{ \mathbf{x} : \mathbf{x} \in T, \alpha(\mathbf{x}) = i \right\}, \tag{2}$$

where

$$\alpha(\mathbf{x}) = \arg\min_{1\le j\le N} d\left(\mathbf{x}, \mathbf{y}_j\right). \tag{3}$$

Necessary Condition 2: Given the partition, the optimal codewords minimizing average distortion D in (1) should satisfy

$$\mathbf{y}_i = \frac{1}{Card\left(T_i\right)}\sum_{\mathbf{x}\in T_i} \mathbf{x}. \tag{4}$$

The K-means algorithm starts with a set of initial codewords. Given the set of codewords, an optimal partition is obtained using (3). After that, given the optimal partition obtained from the previous step, a set of optimal codewords is computed using (4). This process will be repeated until the convergence of the average distortion D of the VQ is observed.

Although the K-means algorithm is effective, the algorithm can only find the local optimal solution for the VQ design. Its performance is dependent on the initial codewords. It may fall into a poor local optimum when the initial codewords are improperly selected.

An alternative to the K-means algorithm is to employ the basic MA, which uses the generational GA and K-means algorithm for the minimization of D. The generational GA is used for the global search. It may find a near optimal solution for the VQ design. The K-means algorithm then uses the results of the global search as its initial codewords for the subsequent local refinement.

In the basic MA, there are P genetic strings for the genetic operations. Each string r represents a set of N codewords $\{\mathbf{y}_1,\ldots, \mathbf{y}_N\}_r$. Note that these strings are strings of vectors, not strings of binary numbers.

Let $S(k)$ and $D(k)$ denote the set of P strings and the value of current minimum distortion D after the execution of the k - th generation of the basic MA, respectively. Let s* be the current optimum string during the course of genetic operations. In the initial step, we let $D(0) = \infty$, and initialize s^* as null. In addition, we can randomly select vectors from training data as the codewords of strings in $S(0)$.

Suppose that the $(k-1)$ - th iteration is completed, and the execution of the k - th ($k \ge 1$) is to be done. We then perform the following genetic operations and local refinement sequentially on the strings in $S(k-1)$.

Regeneration: Since each string in $S(k-1)$ for the genetic operations is in fact a codebook of VQ, its corresponding D can be computed by using (1). The inverse of D is used as the fitness function for each string. The regeneration process is then conducted by using the roulette-wheel technique. That is, for offspring generation, we spin a simulated biased roulette-wheel whose slots have different sizes proportional to the fitness values of the individual strings.

Once a string has been selected for reproduction, an exact replica of it is made as a regeneration string. This regeneration string will then be used for crossover and mutation. In the algorithm, P regeneration strings are created after the regeneration operation.

Crossover: On each regeneration string r, $\{\mathbf{y}_1, \mathbf{y}_2,\ldots, \mathbf{y}_N\}_r$, one point crossover is applied with probability P_c. Out of the total population, a partner string r', $\{\mathbf{z}_1, \mathbf{z}_2,\ldots, \mathbf{z}_N\}_{r'}$, is randomly chosen. Then a random integer n, between 1 and N, is generated. Both strings are cut into two portions at position n and the portions $\{\mathbf{y}_{n+1},\ldots, \mathbf{y}_N\}$ and $\{\mathbf{z}_{n+1},\ldots, \mathbf{z}_N\}$ are mutually exchanged.

Mutation: Mutation is performed on each codeword of each string with a small probability P_b. Suppose now the string $r = \{\mathbf{y}_1, \mathbf{y}_2,\ldots, \mathbf{y}_N\}_r$ is to be mutated. One of the N codewords, $\mathbf{y}$, is chosen at random. Among the w numbers in $\mathbf{y}$, we also select one number at random. Then a random number, taking the binary values b or $-b$, is generated, and is added to the chosen component.

Local Refinement: We apply the K-means algorithm to each string for the local refinement of the genetic operations. The P strings after the K-means design are then the strings of the set $S(k)$. The average distortion D value of each string in $S(k)$ is computed. Let r^* be the string in $S(k)$ having minimum D value, and D^* be the D value of r^*. We then compare D^* with $D(k-1)$. If D^* is smaller than $D(k-1)$, then $D(k) \leftarrow D^*$, and $s^* \leftarrow r^*$. Otherwise, $D(k) \leftarrow D(k-1)$, and the current optimum string s^* is retained the same. This completes the execution of the k - th generation for the basic MA algorithm.

In the basic MA algorithm, the iteration continues until the

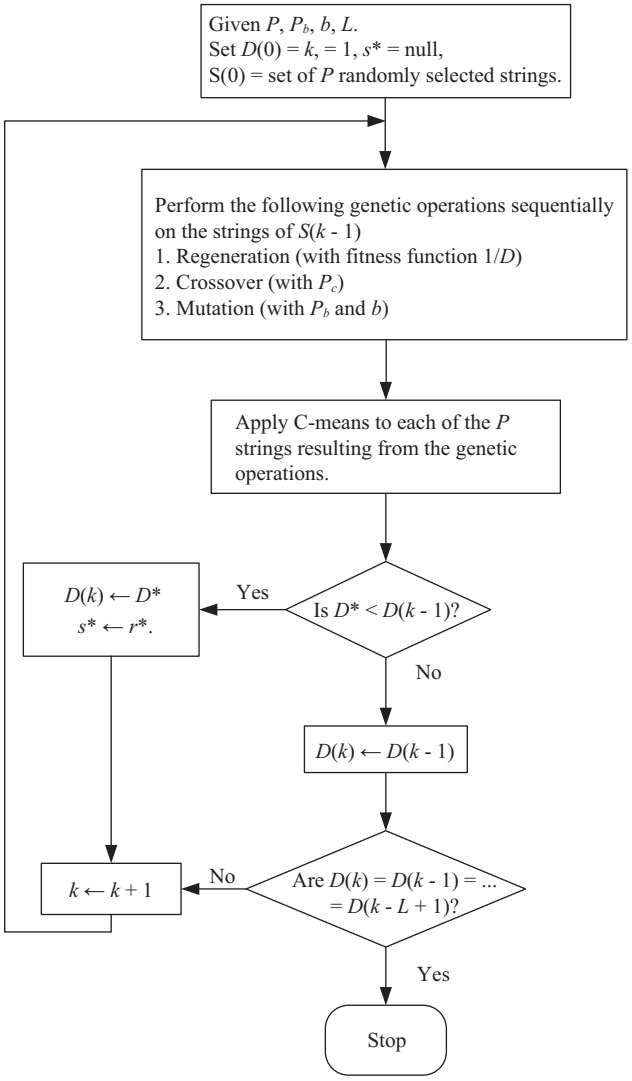

Fig. 1. The flowchart of the basic MA algorithm.

convergence of the sequence $D(k)$. In practice, we stop the design algorithm after the observation of L consecutive iterations yielding identical $D(k)$ value (that is, $D(k) = D(k-1) = \cdots = D(k - L + 1)$). The current optimum string s^* after the completion of basic MA algorithm is then chosen as the desired result. The flowchart of the basic MA algorithm is shown in Fig. 1.

The major drawback of the basic MA is that the K-means algorithm should be applied to every string in the new population for local refinement after the mutation and crossover operations, as illustrated in Fig. 1. From (3), it follows that the K-means algorithm uses the full-search scheme for the partitioning process. The computational complexity of the K-means algorithm therefore is high. When the population size P is large, the basic MA may require long CPU time for finding the optimal solution.

This paper presents a novel steady-state MA algorithm for VQ design. The algorithm has significantly lower computational time as compared with the basic MA while maintaining a comparable performance of distortion rate. In the proposed MA, the steady-state GA is adopted for the global search. The steady-state GA does not use the concept of generation for evolution over populations of candidate solutions. Therefore, it is not necessary to replace all the parent strings by child strings. In fact, at most one parent string is replaced at a time. The replacement process starts by first randomly selecting two parent strings r_1 and r_2 for crossover operation. Assume $r_1 = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}_{r_1}$, and $r_2 = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}_{r_2}$. After the crossover operation, r_1 and r_2 then become

$$\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}_{r_1} \rightarrow \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n, \mathbf{z}_{n+1}, \ldots, \mathbf{z}_N\}_{r_1}$$

$$\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}_{r_2} \rightarrow \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n, \mathbf{y}_{n+1}, \ldots, \mathbf{y}_N\}_{r_2}$$

respectively.

Let c denote the child string of the proposed algorithm. In our design, string c is first obtained by randomly selecting either r_1 or r_2 after the crossover operation. The resulting string c is then mutated and evaluated by the fitness function. If the new child string is inferior to all the parent strings, no parent string will be removed. Otherwise, the parent string with lowest fitness value is replaced by the new child string.

We use the K-means algorithm for the local refinement of child string c after the crossover and mutation operations in the steady-state MA. The average distortion of the new string c produced by the K-means is then evaluated. Let $\underline{r}$ be the string in current S having lowest fitness value (i.e., highest average distortion). Moreover, let $\underline{D}$ be the average distortion of the string $\underline{r}$. If the average distortion of c is less than $\underline{D}$, then S will be updated by replacing $\underline{r}$ with c. Subsequently, all the strings in the updated S are sorted in accordance with their average distortion. The string with highest average distortion then becomes the new $\underline{r}$. This process is repeated until all the parent strings in S survive from the challenges of L consecutive child strings. The parent string in S having the minimum distortion is then selected as the desired VQ codebook. The complete outline of the algorithm is listed below:

Step 0:
Given: P, P_b, b, L.
Initial S = Set of P randomly selected strings.
$\underline{r}$ = string in S having lowest fitness value.
$\underline{D}$ = average distortion of the string $\underline{r}$.

Step 1:
Randomly select two strings r_1 and r_2 from S.

Step 2:
Obtain the child string c by the crossover operation of r_1 and r_2.
Mutate c with P_b and b.

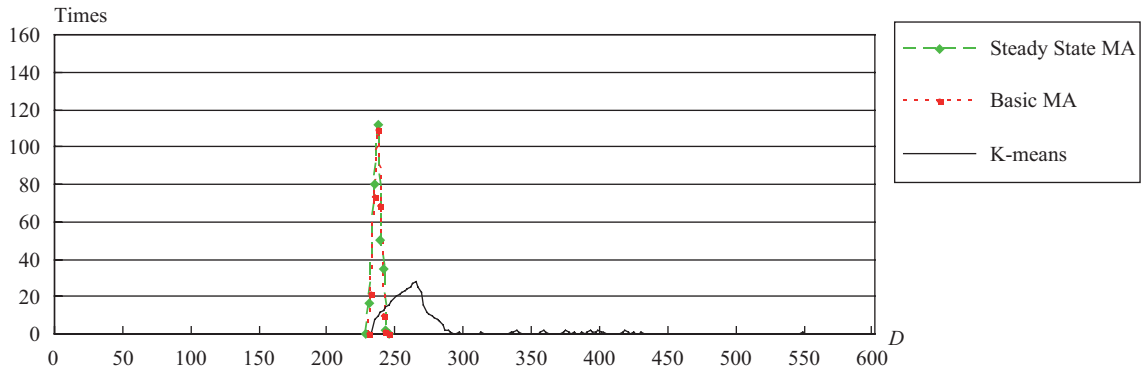

Fig. 3. The distribution of the average distortion for the proposed algorithm, the Basic MA algorithm and the K-means algorithm.

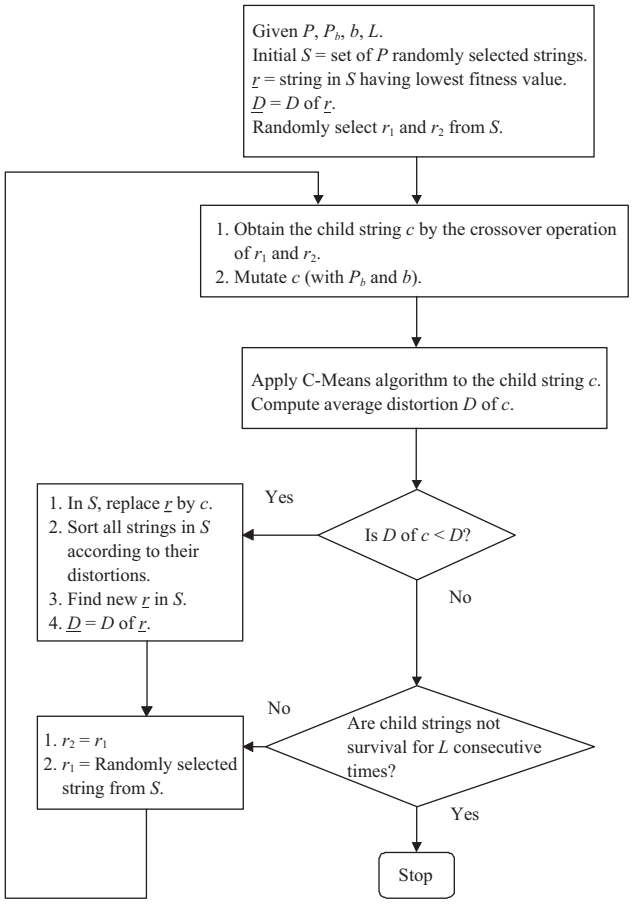

Fig. 2. The flowchart of the steady-state MA algorithm.

Step 3:

Apply K-means algorithm to the child string c.

Compute the average distortion D of c after the K-means algorithm.

Step 4:

If the average distortion D of c is less than D, then

In S, replace $\underline{r}$ by c,

Sort all strings in S according to their distortions,

Find new $\underline{r}$ in S,

Go to Step 1,

else if all the parent strings in S are survival for L consecutive times, then

Select the string having minimum distortion in S as the desired result,

Stop the algorithm.

The flowchart of the steady-state algorithm is shown in Fig. 2. It can be observed from Step 3 of the outline that the K-means algorithm operates only on the string c. It is not necessary for the K-means algorithm to operate over the entire set of genetic strings S in the proposed steady-state MA. In contrast, the K-means algorithm should operate over the entire set $S(k)$ for each iteration k in the basic MA algorithm, as shown in Fig. 1. The proposed algorithm therefore may have lower computational load for the local refinement. The improvement in computational time may be significant when the number of genetic strings P for the design is large.

III. EXPERIMENTAL RESULTS

This section presents some numerical results of the proposed algorithm. All the experiments considered in this section are executed on the Pentium Core 2 Duo processor with 2 Gbytes main memory. Three 512 × 512 images "Lena", "Bridge" and "Boat" are used as the training data for the VQ design.

Figure 3 shows the distribution of the average distortion of steady-state MA for 200 independent runs. Each run starts with different set of genetic strings randomly selected from training images. The number of genetic strings in the experiment is $P = 32$. Each genetic string contains $N = 32$ codewords. The dimension of codewords is $w = 2 \times 2$. The mutation probability is set to be $P_b = 0.01$. The distribution of basic MA for 200 independent runs with the same P, N, w and P_b is also shown in Fig. 3. Moreover, the distribution of K-Means for 200 independent runs with the same N and w is included in Fig. 3 for comparison purpose. The initial codewords of basic MA and K-means algorithm are also randomly selected.

Table 1. The average CPU time and distortion of the proposed algorithm and the basic MA algorithm for various population sizes.

P	Steady State MA		Basic MA	
	CPU Time (Min.)	Ave. Dist.	CPU Time (Min.)	Ave. Dist.
8	3.4	238.74	9.17	241.13
16	7.19	235.26	25.11	236.82
32	15.1	232.84	50.52	235.49
64	31.53	231.17	131.66	234.62
128	62.81	230.31	256.28	232.18

From Fig. 3, it can be observed that the K-means algorithm has a broad distribution of local optima. Results between the best and worst cases differed by more than 100%. On the other hand, from Fig. 3, we see that the distribution of distortion of the steady-state MA has a better concentration. The worst case of steady-state MA has distortion D = 242. Only 6.0% of the distortion of VQs designed by K-means algorithm are lower than that of the worst case of the K-means algorithm. The best case of the steady-state MA has D =229. The difference between the worst and best cases is only 13.

We can also observe from Fig. 3 that both the steady-state GA and basic MA have similar distribution of distortion. Nevertheless, the average CPU time per each run of the proposed algorithm is 15 mins. In contrast, the average CPU time of the basic MA is 50 mins. The CPU time of the proposed algorithm is only 30% of that of the basic MA.

Table 1 shows the average CPU time and distortion of the proposed algorithm and the basic MA algorithm for various population sizes. In this experiment, we fix the number of codewords in each string N as 32, the dimension of codewords w as 2×2, and the mutation probability as P_b = 0.01. The average CPU time and distortion for each P in the table are obtained by averaging the results of 100 independent runs. From the table, we observe that as P increases, the CPU time of the basic MA increases significantly. However, the proposed algorithm remains low CPU time even when P becomes large. In addition, both the steady-state MA and basic MA having comparable performance give the same P.

Finally, given N = 32 and $w = 2 \times 2$, the average distortions of the proposed algorithm for various pairs of (P, P_b) are shown in Table 2. Similar to Table 1, the average distortion for each pair of (P, P_b) in the table is obtained by averaging the results of 100 independent runs. From the table, it can be concluded that the variation in performance for different set of parameters is small. All these facts demonstrate the effectiveness of the proposed algorithm.

Table 2. The average distortion of the proposed algorithm for various pairs of (P, P_b).

	P_b = 0.01	P_b = 0.05	P_b = 0.1	P_b = 0.15	P_b = 0.2
P = 8	238.74	241.18	245.91	244.32	246.35
P = 16	235.26	239.31	243.71	243.86	244.88
P = 32	232.84	238.3	242.26	242.44	244.17
P = 64	231.17	237.47	240.52	240.50	241.74

IV. CONCLUDING REMARKS

Experimental results show that our steady-state MA has low distortion and low CPU time for VQ design. Based on the same population size, the steady-state MA is able to save more than 70% of the CPU time of the basic MA. In addition, both the steady-state MA and basic MA have similar distribution of performance over 200 independent runs. The distribution of the proposed algorithm has a better concentration as compared with the basic K-means algorithm. This result reveals that the proposed algorithm is insensitive to the selection of initial codewords. The proposed algorithm therefore is beneficial for VQ training where both the computational complexity and distortion are important concerns.

REFERENCES

1. Areibi, S., Moussa, M., and Abdullah, H., "A comparison of genetic/ memetic algorithms and heuristic searching," *Proceedings of the 2001 International Conference on Artificial Intelligence*, Las Vegas, Nevada, pp. 660-666 (2001).
2. Coe, S., Areibi, S., and Moussa, M., "A hardware memetic accelerator for VLSI circuit partitioning," *Computers and Electrical Engineering*, Vol. 33, pp. 233-248 (2007).
3. Eiben, A. E. and Smith, J. E., *Introduction to Evolutionary Computing*, Springer, Berlin (2003).
4. Gersho, A. and Gray, R. M., *Vector Quantization and Signal Compression*, Kluwer, Boston (1992).
5. Goldberg, D. E. and Deb, K., "A comparative analysis of selection schemes used in genetic algorithms," In: Rawlins, G. (Ed.), *Fundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, pp. 69-93 (1991).
6. Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y., "An efficient C-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 7, pp. 881-892 (2002).
7. Lin, T. K., Li, H. Y., Hwang, W. J., Ou, C. M., and Weng, S. K., "Genetic vector quantizer design on reconfigurable hardware," *Lecture Notes in Computer Science*, Vol. 5361, pp. 473-482 (2008).
8. Merz, P. and Freisleben, B., "Genetic algorithms for binary quadratic programming," *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, pp. 417-424 (1999).
9. Radcliffe, N. J. and Surry, P. D., "Formal memetic algorithms," *Lecture Notes in Computer Science*, Vol. 865, pp. 1-16 (1994).
10. Rasheed, K. and Davisson, B. D., "Effect of global parallelism on the behavior of a steady state genetic algorithm for design optimization," *Proceedings of the Congress on Evolutionary Computation*, Vol. 1, pp. 541 (1999).
11. Scheunders, S., "A genetic C-means clustering algorithm applied to color image quantization," *Pattern Recognition*, Vol. 30, No. 6, pp. 859-866 (1997).
12. Srinivas, M. and Patnaik, L. M., "Genetic algorithm: A survey," *IEEE Computer*, Vol. 27, pp. 17-26 (1994).