

Hybrid Genetic Algorithm for Design of Robust Communication Systems

Chien-Min Ou, Jing-Jhih Chen
 Department of Electronics Engineering,
 Ching Yun University, Chungli, Taiwan 320, R.O.C.
 Email: {cmou, m9411014} @cyu.edu.tw

Wen-Jyi Hwang¹
 Graduate Institute of Computer Science and Information Engineering,
 National Taiwan Normal University, Taipei, Taiwan 117, R.O.C.
 Email: whwang@csie.ntnu.edu.tw

Abstract—A novel hybrid genetic algorithm (GA) for jointly optimizing source and channel codes is presented in this paper. The algorithm first uses GA for the coarse search of source and channel codes. An iterative search is then followed for the refinement of the coarse search. The hybrid GA enhances the robustness of the design of source and channel codes. The distributed GA scheme can also be used in conjunction with the proposed hybrid GA algorithm for further performance improvement.

Index Terms—Vector Quantization, Genetic Algorithm, Error Correct Coding.

I. INTRODUCTION

The objective of designing a robust communication system is to minimize the end-to-end average distortion of the system over a noisy channel. The basic techniques for the design can be classified into three classes: the channel-optimized source coding, the source-optimized channel coding, and the combination of these two classes. The channel-optimized source coding techniques design source codes of the communication system optimally matched to a given noisy channel. Typical examples are the channel-optimized vector quantization (COVQ) [2] and its variants. The source-optimized channel coding techniques usually construct unequal error protection (UEP) schemes best matched to a given source code. Some variable-rate channel codes such as rate-compatible punctured convolutional codes (RCPCs) [5] have been found to be effective for the implementation of UEP. The application of RCPCs to the UEP is realized by a bit allocation process, which determines the degree of error protection to different locations of the binary channel indices representing VQ codewords.

The combination of channel-optimized source coding and source-optimized channel coding may further improve the performance of the joint design. An iterative

procedure optimizing source code and channel code one at a time has been employed to realize this combination [4]. Although the iterative scheme is effective, it has two major drawbacks. First, its source code at each iteration is designed using the COVQ, which usually falls into a poor local optimum. Therefore, the results of the iterative scheme may also be a local optimal solution. Second, each iteration of the scheme consists of the design of both source and channel codes. Its computational complexity is higher than the algorithms designing only source or channel codes. In addition, the full-search bit allocation scheme is used for the UEP, which may require high computational time for long binary channel indices [7].

One way to prevent the joint design from getting trapped in a poor local optimum is to adopt the stochastic optimization. The simulated annealing (SA) has been found to be an effective stochastic optimization technique for the design of source codes [15]. However, the annealing schedule, the rate at which the temperature is lowered, should be carefully selected in the algorithm. The schedule achieving global optimum requires tremendous computational complexities [3], and is not realistic in many applications. Other schedules that accelerate the cooling process can reduce the computation time at the expense of possible degradation in performance.

In addition to the SA, an extensively used algorithm for the stochastic optimization is the genetic algorithm (GA) [13]. Inspired by biological evolution, the GA has been successfully used for global search. The basic 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. Moreover, crossover and mutation are employed to obtain better strings. The mutation operator changes individual elements of a string, and the crossover operation interchanges parts between strings.

While the GA is good at coarse search over the entire solution space, it may not be suited for fine tuning search results which are close to optimal. To eliminate this

¹To whom all correspondence should be sent

drawback, various hybrid GAs (or memetic algorithms) combining the GA with local search have been proposed. In the hybrid GAs, local improvement operations for fine tuning are immediately applied to each genetic strings after reproduction, mutation and crossover operations. Superior performance over pure GA has been found for various applications such as VLSI design [1], traveling salesman problem [12], binary quadratic programming [9] and VQ design [8].

In light of the facts stated above, we employ a new hybrid GA technique for the joint design. In the new technique, the pure GA is used for the concurrent coarse search of source and channel codes. Based on the results of coarse search, the COVQ is then adopted for the local improvement of source codes. Note that it may not be necessary to fine tune the channel codes because the coarse search based on pure GA has comparable performance to that of the full search [7]. The concurrent design requires only one GA search so that the algorithm may have lower computational complexity as compared with its iterative counterpart. The cost function for the concurrent search is a weighted sum of the average distortion and transmission rate. It therefore may have superior rate-distortion performance over its iterative counterpart where the GA searches only take average distortion and transmission rate into account one at a time.

The proposed algorithm can be extended by incorporating the distributed operations for further performance improvement. In this extension, the GA strings are partitioned into a number of groups, called islands. The hybrid genetic operations for each island are performed independently. Some genetic strings of each island may migrate to the other islands after a pre-specified number of generations. This process permits more robust coarse search; thereby attaining superior rate-distortion performance over the hybrid GA without distributed operations. Numerical results show that the algorithm can be an effective alternative for the design of robust communication systems over noisy channels.

II. PROBLEM FORMULATION

Consider a full-search VQ with N codewords $\mathbf{y}_1, \dots, \mathbf{y}_N$. Each codeword \mathbf{y}_i is represented by a binary index c_i with length n , where $n = \log N$. Let $c_i(m), m = 1, \dots, n$, be the m -th bit of c_i . Suppose the noisy channel is a binary symmetric channel (BSC) with bit error rate (BER) ε . In addition, the RCPC is used for the error correction of binary indices. The set of channel code rates from Table I in [5] (denoted by C) are used to obtain our RCPC candidates. We represent each candidate code by a vector with dimension n where the m -th element in the vector is the channel code rate applied to $c_i(m)$. For example, suppose $n=3$. The RCPC code $\{1/2, 2/3, 2/3\}$ applies the convolutional code with rate $1/2$ to $c_i(1)$, and the convolutional code with rate $2/3$ to $c_i(2)$ and $c_i(3)$. The average transmission rate of the VQ is defined as the average number of bits representing each source vector after the channel encoding process.

Consequently, for a RCPC code $\{s_1, \dots, s_n\}$, the average transmission rate is given by $\sum_{m=1}^n \frac{1}{s_m}$.

Let $P_{k|i}$ be the probability that the binary index c_i delivered by VQ encoder is received as c_k by the VQ decoder because of channel errors. We call $P_{k|i}, i, k = 1, \dots, N$, the index crossover probabilities, which are functions of RCPC and the bit error rate (BER) of the BSC. Given source codewords and index crossover probabilities, the average end-to-end distortion, D , is given by

$$D = \frac{1}{\omega t} \sum_{j=1}^t \sum_{k=1}^N P_{k|\alpha(\mathbf{x}_j)} d(\mathbf{x}_j, \mathbf{y}_k). \quad (1)$$

where ω is the vector dimension, $\{\mathbf{x}_j\}_{j=1}^t$ are source vectors, $\alpha(\mathbf{x}_j)$ is the source encoder, and $d(\mathbf{u}, \mathbf{v})$ is the squared distance between vectors \mathbf{u} and \mathbf{v} . The goal of joint design is then equivalent to the following optimization problem:

$$\begin{aligned} \min_{\substack{(\mathbf{y}_1, \dots, \mathbf{y}_N) \\ (s_1, \dots, s_n)}} \frac{1}{\omega t} \sum_{j=1}^t \sum_{k=1}^N P_{k|\alpha(\mathbf{x}_j)} d(\mathbf{x}_j, \mathbf{y}_k), \\ \text{subject to } \sum_{m=1}^n \frac{1}{s_m} \leq R, \end{aligned} \quad (2)$$

where R is the constraint on the average transmission rate. To solve this problem, the Lagrangian method with cost function J can be used, where

$$J = \frac{1}{\omega t} \sum_{j=1}^t \sum_{k=1}^N P_{k|\alpha(\mathbf{x}_j)} d(\mathbf{x}_j, \mathbf{y}_k) + \lambda \sum_{m=1}^n \frac{1}{s_m}, \quad (3)$$

and $\lambda > 0$ determines the resulting transmission rate after the optimization.

III. THE ALGORITHMS

In this section, five joint design algorithms are presented: the GA-based COVQ (G-COVQ) algorithm, the GA-based UEP (G-UEP) algorithm, the iterative combination of G-COVQ and G-UEP (GA-based iterative) algorithm, the GA-based concurrent design algorithm and distributed GA-based concurrent algorithm[6,10-11,14].

A. G-COVQ Algorithm

We first introduce the COVQ algorithm, which is the basic channel-optimized source coding technique. In the COVQ design, we assume that the BER ε of the BSC channel and RCPC rates $\{s_1, \dots, s_n\}$ are fixed. The index crossover probabilities $P_{k|i}, i, k = 1, \dots, N$, thereby are also fixed. Consequently, the minimization of J given in eq. (3) is simply equivalent to the minimization of D . Hence, the objective of the COVQ design can be stated as finding a set of VQ codewords $\mathbf{y}_k, k = 1, \dots, N$, minimizing D . It can be shown that, given codewords $\mathbf{y}_k, k = 1, \dots, N$, the optimal source encoder α minimizing D should satisfy

$$\alpha(\mathbf{x}_j) = \arg \min_{1 \leq l \leq N} \sum_{k=1}^N P_{kl} d(\mathbf{x}_j, \mathbf{y}_k). \quad (4)$$

In addition, given α , the optimal codewords \mathbf{y}_k , $k=1, \dots, N$, minimizing D can be evaluated as

$$\mathbf{y}_k = \frac{\sum_{j=1}^t P_{k/\alpha(\mathbf{x}_j)} \mathbf{x}_j}{\sum_{j=1}^t P_{k/\alpha(\mathbf{x}_j)}}. \quad (5)$$

The COVQ algorithm is based on an iterative procedure where source encoder α and codewords $\mathbf{y}_k, k=1, \dots, N$ are optimized one at a time using eqs. (4) and (5), respectively. The major disadvantage of the COVQ is that its performance is sensitive to the selection of initial codewords, which can be solved by the G-COVQ algorithm.

Suppose there are G strings in the algorithm. Each string $\mathbf{g} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}_{\mathbf{g}}$ is a set of VQ codewords. Let $\mathcal{G}(q)$ be the set of G strings after the execution of the q -th evolution, where each evolution consists of regeneration, crossover, mutation and COVQ optimization operations.

Let \mathbf{g}^* be the *current optimal* string during the course of the GA. We set the initial \mathbf{g}^* as null. In addition, the VQ codewords in $\mathcal{G}(0)$ are formed by randomly selecting source vectors in $\{\mathbf{x}_j\}_{j=1}^t$. Now, suppose the $(q-1)$ -th evolution is completed, and the execution of the q -th evolution is to be done. We then perform the following genetic operations sequentially on the strings in $\mathcal{G}(q-1)$.

Regeneration of G-COVQ: Since each string in $\mathcal{G}(q-1)$ contains VQ codewords, their corresponding D can be computed using eq. (1). The inverse of D is used as a fitness function for each string. The regeneration process is then conducted using the roulette-wheel technique. There are G regeneration strings created after the regeneration operation.

Crossover of G-COVQ: On each regeneration string \mathbf{g} , $\{\mathbf{y}_1, \dots, \mathbf{y}_N\}_{\mathbf{g}}$, the crossover operation is applied with probability P_c . Out of the total population, a partner string \mathbf{g}' , $\{\mathbf{y}'_1, \dots, \mathbf{y}'_N\}_{\mathbf{g}'}$, is randomly chosen. Then an integer random number b between 1 and n is generated. Both strings are cut into two portions at position b , and the portions $\{\mathbf{y}_{b+1}, \dots, \mathbf{y}_N\}$ and $\{\mathbf{y}'_{b+1}, \dots, \mathbf{y}'_N\}$ are mutually exchanged.

Mutation of G-COVQ: For each string \mathbf{g} , mutation is performed on each element of $\mathbf{y}_k, k=1, \dots, N$, with a small probability P_m . Suppose \mathbf{y}_k is determined to be mutated. One of the w components of \mathbf{y}_k is then selected at random. A random number, taking binary values b or $-b$, is generated, and is added to the selected component.

COVQ optimization of G-COVQ: After the regeneration, crossover and mutation operations, the COVQ algorithm is applied to each string \mathbf{g} . The initial codewords and index crossover probabilities for the

COVQ design are obtained from the VQ codewords and RCPC rates of that string, respectively. The resulting codewords after the COVQ design will replace the original VQ codewords in that string. The G strings after the COVQ design are then the strings of the set $\mathcal{G}(q)$.

Test for Convergence of G-COVQ: After the completion of the COVQ optimization, The D value of each string in $\mathcal{G}(q)$ is computed. Let \mathbf{g}^{**} be the string in $\mathcal{G}(q)$ having minimum D , and D^{**} be the D of \mathbf{g}^{**} . We then compare D^{**} with D^* , the D of \mathbf{g}^* (D^* is initialized as ∞). If D^{**} is smaller than D^* , then $D^* \leftarrow D^{**}$ and $\mathbf{g}^* \leftarrow \mathbf{g}^{**}$. Otherwise, both D^* and \mathbf{g}^* are retained the same. This completes the execution of q -th evolution of our genetic programming algorithm. In the algorithm, the evolution continues until the observation of I consecutive evolutions yielding identical D^* value.

B. G-UEP Algorithm

The G-UEP algorithm can be used to reduce the computational complexity of the UEP [7]. Let S_i be the cluster such that $S_i = \{\mathbf{x}_j : \alpha(\mathbf{x}_j) = i\}$ and \mathbf{z}_i be the centroid of S_i . We can rewrite eq. (1) as

$$D = \frac{1}{\omega t} \sum_{j=1}^t d(\mathbf{x}_j, \mathbf{y}_{\alpha(\mathbf{x}_j)}) + \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^N P_{ki} d(\mathbf{y}_i, \mathbf{y}_k). \quad (6)$$

Note that, the first term in eq. (6) depends only on the VQ codewords \mathbf{y}_i and source vectors \mathbf{x}_j . Therefore, this term does not change as a function of RCPC code. The optimal RCPC code only minimizes the second term of eq. (6):

$\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^N P_{ki} d(\mathbf{y}_i, \mathbf{y}_k)$. Since the first term requires higher computational complexity, given a set of VQ codewords $\{\mathbf{y}_1, \dots, \mathbf{y}_n\}$, the objective function J in eq. (3) for UEP design can be simplified into

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^N P_{ki} d(\mathbf{y}_i, \mathbf{y}_k) + \lambda \sum_{m=1}^n s_m^{-1}, \quad (7)$$

The problem of the UEP therefore is equivalent to find a set of RCPC rates $\{s_1, \dots, s_n\}$ minimizing the cost function L given in eq. (7) for a fixed set of VQ codewords. Suppose there are G strings. Each string $\mathbf{s} = \{s_1, \dots, s_n\}_{\mathbf{s}}$ is a set of RCPC rates. Let $\mathcal{S}(q)$ be the set of G strings after the execution of the q -th evolution, where each evolution consists of regeneration, crossover and mutation operations. Let \mathbf{s}^* be the *current optimal* string during the course of the GA, and L^* be its L value. We set the initial \mathbf{s}^* and L^* as null and ∞ , respectively. In addition, the strings in $\mathcal{S}(0)$ is formed by randomly selecting channel rates in \mathcal{C} . Now, suppose the $(q-1)$ -th evolution is completed, and the execution of the q -th iteration is to be done. We then perform the following genetic operations sequentially on the strings in $\mathcal{S}(q-1)$.

Regeneration of G-UEP: Each string in $\mathcal{S}(q-1)$ in fact is a RCPC code. Hence, their corresponding L can be computed using eq. (7). The inverse of L is used as a fitness function for each string. The regeneration process

is then conducted in the manner similar to that of G-COVQ. There are G regeneration strings created after the regeneration operation.

Crossover of G-UEP: This process is similar to that of the G-COVQ with crossover probability P_c for each string s .

Mutation of G-UEP: Mutation is performed on each element $s_m, m=1, \dots, n$, of each string with a small probability P_m . Suppose s_m is determined to be mutated, then a rate selected at random from \mathcal{C} is used to replace s_m .

Test for Convergence of G-UEP: The G strings after these operations are the strings of $\mathcal{S}(q)$. The L value of each string in $\mathcal{S}(q)$ is computed. Based on these L values, the s^* and L^* can be updated in the way similar to that for updating g^* and D^* in the G-COVQ. This completes the execution of q -th evolution of G-UEP algorithm. In the G-UEP algorithm, the evolution continues until the observation of I consecutive evolutions yielding identical L^* value.

C. GA-Based Iterative Algorithm

In the iterative algorithm, each iteration executes the G-COVQ and G-UEP sequentially. Let $\{y_1^f, \dots, y_N^f\}$ and $\{s_1^f, \dots, s_n^f\}$ be the set of VQ codewords and RCPC rates after the design of the f -th iteration, respectively. Now, suppose the $(f-1)$ -th iteration is completed, and the design of the f -th iteration is to be done. Each iteration contains two steps, which correspond to G-COVQ and G-UEP design, respectively.

Step 1: Given $\{s_1^{f-1}, \dots, s_n^{f-1}\}$, the objective at this step is to design $\{y_1^f, \dots, y_N^f\}$ using the G-COVQ algorithm. The set of RCPC rates $\{s_1^{f-1}, \dots, s_n^{f-1}\}$ is used to determine the index crossover probabilities $P_{k/i}, i, k = 1, \dots, N$, for the computation of D given in eq. (1). The VQ codewords $\{y_1^f, \dots, y_N^f\}$ at the iteration f is then set to be the final current optimal string g^* after the completion of G-COVQ.

Step 2: Using the G-UEP, this step finds the RCPC rates $\{s_1^f, \dots, s_n^f\}$ best matched to the VQ codewords $\{y_1^f, \dots, y_N^f\}$ designed at the previous step. The VQ codewords are used to compute the first term in L shown in eq. (7) (i.e., $\frac{1}{N} \sum_{k=1}^N \sum_{i=1}^N P_{k/i} d(y_i, y_k)$) for the execution of the G-UEP. The RCPC rates $\{s_1^f, \dots, s_n^f\}$ at the iteration f is then set to be the final current optimal string s^* after the completion of G-UEP.

Test for Convergence of the Iterative Algorithm: Let J^f be the value of J after the completion of the f -th iteration. Since the minimization of D and L are equivalent to the minimization of J in the G-COVQ and G-UEP, the iteration algorithm will continue until the convergence of the sequence $\{J^f\}$.

D. GA-based Concurrent Algorithm

The GA-based concurrent algorithm can attain both high performance and low computational complexity for the joint design. In the algorithm, each string g in the algorithm can be divided into two segments: the VQ codewords segment $\{y_1, \dots, y_N\}_g$ and the RCPC rates segment $\{s_1, \dots, s_n\}_g$. Let $\mathcal{G}(q)$ be the set of G strings after the execution of the q -th evolution, where each evolution consists of regeneration, crossover, mutation and COVQ optimization operations. Let g^* be the *current optimal* string during the course of the GA and J^* be its J value. We set the initial g^* as null, and initial $J^* = \infty$. In addition, the VQ codewords and RCPC rates of each string in $\mathcal{G}(0)$ are formed by randomly selecting source vectors and channel rates in $\{x_j\}_{j=1}^N$ and \mathcal{C} , respectively. Now, suppose the $(q-1)$ -th evolution is completed, and the execution of the q -th evolution is to be done. We then perform the following genetic operations sequentially on the strings in $\mathcal{G}(q-1)$.

Regeneration of GA-based concurrent algorithm: We use the inverse of J given in eq. (3) as the fitness function for each string in $\mathcal{G}(q-1)$. The regeneration process is then conducted in the manner similar to that of the G-COVQ and G-UEP. There are G regeneration strings created after the regeneration operation.

Crossover of GA-based concurrent algorithm: On each regeneration string g the crossover operation is applied with probability P_c . Out of the total population, a partner string g' is randomly chosen. Then two integer random numbers b_1 (between 1 and N) and b_2 (between 1 and n) are generated. The VQ codewords segment and RCPC rates segment of both strings are cut into two portions at positions b_1 and b_2 , respectively. Portions of each segment of strings g and g' are mutually exchanged. The resulting strings are then given by

$$g = \{y_1, \dots, y_{b_1}, y'_{b_1+1}, \dots, y'_N, s_1, \dots, s_{b_2}, s'_{b_2+1}, \dots, s'_n\}_g,$$

$$g' = \{y'_1, \dots, y'_{b_1}, y_{b_1+1}, \dots, y_N, s'_1, \dots, s'_{b_2}, s_{b_2+1}, \dots, s_n\}_{g'}.$$

Mutation of GA-based concurrent algorithm: For each string g , mutation is performed on each element of $y_k, k = 1, \dots, N$, and $s_m, m = 1, \dots, n$, with a small probability P_m . Suppose y_k is determined to be mutated. One of the ω components of y_k is then selected at random. A random number, taking binary values b or $-b$, is generated, and is added to the selected component. For each s_q determined to be mutated, a rate is first selected at random from \mathcal{C} , and s_q is then replaced by the rate.

COVQ optimization of GA-based concurrent algorithm: After the regeneration, crossover and mutation operations, the COVQ algorithm is applied to each string g . The initial codewords and index crossover probabilities for the COVQ design are obtained from the VQ codewords and RCPC rates of that string, respectively. The resulting codewords after the COVQ design will replace the original VQ codewords in that string. The G strings after the COVQ design are then the strings of the set $\mathcal{G}(q)$.

Test for Convergence of GA-based concurrent algorithm: After the completion of the COVQ optimization, the J value of each string in $\mathcal{G}(q)$ is computed. The new g^* and J^* can then be obtained in the way similar to that for updating g^* and D^* in the G-COVQ. This completes the execution of q -th evolution of GA-based concurrent algorithm. In the algorithm, the evolution continues until the observation of l consecutive evolutions yielding identical J^* value.

E. Distributed GA-based Concurrent Algorithm

The concurrent GA schemes can be extended by incorporating the distributed operations for enhancing the robustness of the design of source and channel codes. To perform the distributed GA operations, the GA strings are partitioned into M groups, called islands. In each island, the genetic strings are optimized using the GA-based concurrent algorithm presented in the previous subsection independently. Some genetic strings of each island may migrate to the other islands after a pre-specified number of generations, called migration interval T . The process is continued until the strings of all islands are converged. A complete pseudo code for the distributed GA is as follows.

Distributed GA Algorithm

- Step 0: Given integers $M > 0, T > 0, k > 0$ and a real number $1 > p > 0$.
- Step 1: Divide genetic strings into M groups, where each group is called an island.
- Step 2:

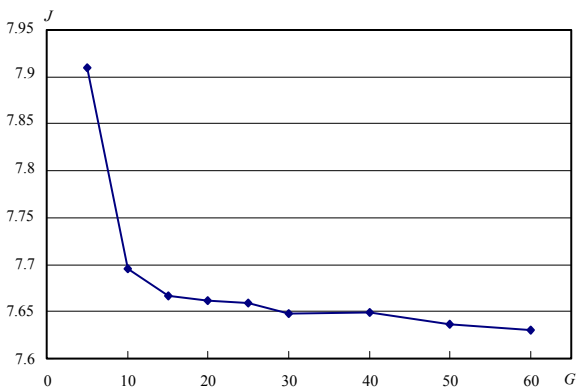


Figure 1. The dependence of the GA-based concurrent algorithm on the population size G .

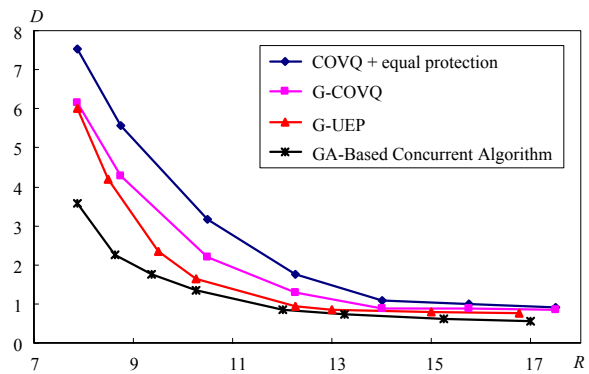


Figure 2. The rate-distortion performance of COVQ, G-COVQ, G-UEP and GA-based concurrent algorithms.

For $j = 1$ to M loop .

- 2.1: Apply GA-based concurrent algorithm to island j for T generations.
- 2.2: End loop.

Step 3:

- If all the islands converge, then stop.
- Otherwise, go to step 4.

Step 4:

- For $j = 1$ to M loop .
- 4.1: Determine randomly whether the string migration for the island j is necessary with probability p .
- 4.2: If no migration is necessary, go to step 4.4.
- 4.3: Let U_j be a set of k strings randomly selected from island j . Swap U_j and U_i , where i is also randomly selected.
- 4.4: End loop.

Step 5: Go to step 2.

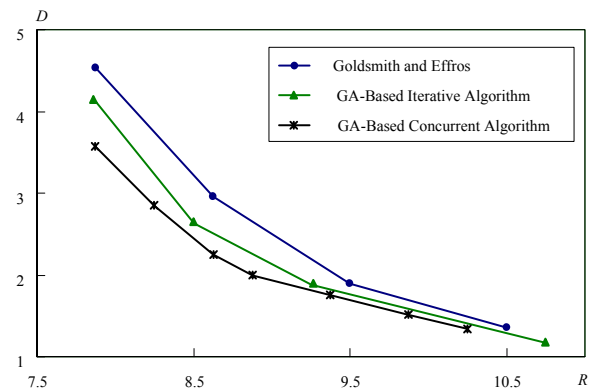


Figure 3. The rate-distortion performance of Goldsmith and Effros, GA-based iterative and GA-based concurrent algorithms.

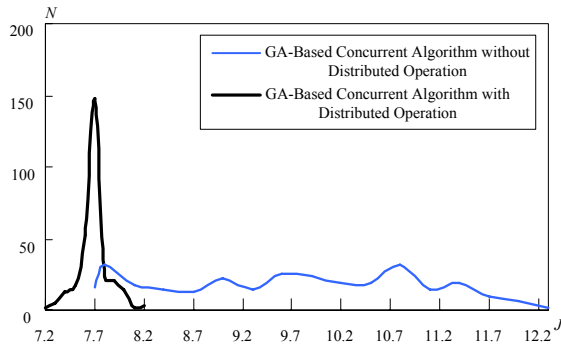


Figure 4. The distribution of the J values of the GA-based concurrent algorithm with and without distribution operations.

IV. SIMULATION RESULTS

This section presents some simulation results of the GA-based concurrent algorithm for the joint design of source and channel codes. The COVQ, G-COVQ, G-UEP, GA-based iterative algorithm and the iterative algorithm proposed by Goldsmith and Effros [4] (termed Goldsmith- Effros algorithm) are also implemented for comparison purpose. The vector dimension is $w=8$ for all the experiments. The BER of the BSC channel is 0.01. The Gauss-Markov sequences with $\rho=0.9$ are used for both training and performance measurement. The total number of samples of the sequences is 84000.

Figure 1 elaborates the dependence of the GA-based concurrent algorithm on the population size G for $\lambda=1$. The number of codewords is $N=32$. Each sample point in the figure is an average value over 100 independent executions using randomly chosen initial genetic strings. From the figure, we observe that the average J decreases as G increases. However, the reduction becomes negligible when $G \geq 15$. Since the computational time grows with G , we choose $G=15$ for the subsequent experiments for attaining low computational complexity.

Table 1 compares the GA-based concurrent algorithm

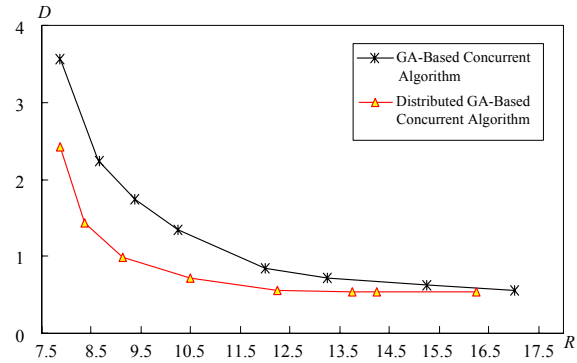


Figure 5. The performance of the GA-based concurrent algorithm and distributed GA-based concurrent algorithm for the BSC with $\epsilon=0.05$.

with its major counterpart, the GA-based iterative algorithm. The comparison includes the mean, minimum, maximum and standard deviation values of J from 100 independent trials of each algorithm for various λ values. It can be observed from the table that the GA-based concurrent algorithm has lower mean, minimum, maximum and standard deviation values for each λ as compared with the GA-based iterative algorithm. This implies that the GA-based concurrent algorithm provides robust solutions with superior performance. The GA-based iterative algorithm does not perform well because it optimizes source and channel codes one at a time iteratively. Poor source codes obtained from the first step of the algorithm will be used to design the channel codes at the second step. This may result in a poor local optimum solution.

In addition to evaluating the cost J for the joint design problem, we also compare the rate distortion performance of the GA-based concurrent algorithm with that of other techniques, as shown in Figures 2 and 3. To achieve meaningful comparisons, all the algorithms have the same number of VQ codewords $N=128$. It is not surprising to observe from Figure 2 that the GA-based concurrent algorithm significantly outperforms the COVQ, G-COVQ and G-UEP algorithms, which only

TABLE I
THE MEAN, MINIMUM, MAXIMUM AND STANDARD DEVIATION VALUES OF COST J FROM 100 INDEPENDENT TRIALS OF THE GA-BASED CONCURRENT ALGORITHM AND GA-BASED ITERATIVE ALGORITHM FOR VARIOUS λ VALUES.

λ	GA-based-Concurrent				GA-based Iterative			
	min	max	mean	variance	min	max	mean	variance
1	7.60	7.99	7.69	0.05	10.61	11.61	10.93	0.19
5	30.42	31.74	30.63	0.13	41.50	49.03	43.55	1.14
10	58.53	59.77	58.76	0.20	70.58	87.89	76.38	3.14
15	86.64	91.72	87.00	0.73	91.27	129.78	103.88	5.35

TABLE II
THE AVERAGE CPU TIME (MEASURED ON 1.6G HZ PENTIUM IV) OF VARIOUS ALGORITHMS FOR THE EXPERIMENTS SHOWN IN FIGURE 2 AND 3.

Algorithm	G-COVQ	GA-based Iterative	GA-based Concurrent	Goldsmith and Effros
CPU Time	1.9 hrs	7.3 hrs	2.1 hrs	60.3 hrs

design either a source code or a channel code. In addition, as illustrated in Figure 3, the GA-based concurrent algorithm also has superior rate-distortion performance over the GA-based iterative algorithm and Goldsmith-Effros algorithm [4], which design both the source and channel codes iteratively. The superiority of the GA-based concurrent algorithm over the GA-based iterative algorithm observed in the figure is consistent with the results shown in Table 1, where the concurrent algorithm attains lower cost for the optimization.

The Goldsmith-Effros algorithm is an iterative algorithm without GA. Therefore, poor results obtained at intermediate iterations will propagate over the subsequent iterations in the algorithm. Moreover, because the Goldsmith-Effros algorithm uses the full-search scheme for finding the channel codes, it has high computational complexity. Table 2 shows the computational complexity of various algorithms for the experiments shown in Figure 2 and 3, where the computational complexity of each algorithm is defined as the average CPU time required for the execution of that algorithm. It can be observed from the table that the average CPU time of the GA-based concurrent algorithm in the experiments is only 2.1 hrs, which is significantly lower than that of the Goldsmith-Effros algorithm (60.3 hrs). In addition, our novel joint source/channel codes design scheme has CPU time comparable to that of the G-COVQ algorithm, which optimizes source codes only. All these facts demonstrate the effectiveness of the GA-based concurrent algorithm.

All the numerical results shown above are based on the GAs without distributed operations. To demonstrate the improvement made by the distributed operations, Figure 4 shows the distribution of the J values of the GA-based concurrent algorithm with and without distributed operations. The λ value for optimization is given by ε . The distributions are obtained from independent applications of each concurrent algorithm 1000 times using randomly chosen initial genetic strings. For the implementation of the distributed GA, the number of islands M and the migration interval T are set to be 3 and 10, respectively. From the figure, we can see that the concurrent algorithm with distributed GA has better concentration. On the contrary, poor initial genetic strings may result in slightly inferior local optimum for the concurrent design without distributed GA. Finally, Figure 5 shows the rate-distortion performance of the concurrent design with and without distributed GA. It can be concluded from the figure that the employment of the distributed GA can effectively enhance the performance

of the GA-based concurrent algorithm for the joint design of the source and channel codes.

V. CONCLUSION

The GA-based concurrent optimization algorithm has been found to be effective for the joint design of source and channel codes. Experimental results show that the algorithm outperforms its iterative counterparts with lower CPU time. In particular, when the number of codewords is 128, the concurrent algorithm without distributed operation only requires 3.48% of CPU time of the Goldsmith-Effros algorithm. In addition, the performance of the concurrent algorithm can be improved further by the employment of distributed operations. The algorithm therefore is for the applications where low computational complexity and/or high performance are desired for the design of robust transmission systems.

REFERENCES

- [1] S. Areibi, M. Moussa and H. Abdullah, "A Comparison of Genetic/Memetic Algorithms and Heuristic Searching," *Proc. International Conference on Artificial Intelligence*, pp.660-666, June 2001.
- [2] N. Farvardin and V. Vaishampayan, "On the performance and complexity of channel optimized vector quantizers," *IEEE Trans. Information Theory*, Vol. 37, pp.155-160, 1991.
- [3] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 6, pp.721-741, 1984.
- [4] A. J. Goldsmith and M. Effros, "Joint design of fixed-rate source codes and multiresolution channel codes," *IEEE Trans. Commun.*, Vol.46, pp. 1301-1311, Oct. 1998.
- [5] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC) codes and their applications," *IEEE Trans. Comm.*, Vol.36, pp.389-400, 1988.
- [6] T. M. Hamdani, A. M. Alimi, F. Karray, "Distributed Genetic Algorithm with Bi-Coded Chromosomes and a New Evaluation Function for Features Selection," *IEEE Congress on Evolutionary Computation*, pp. 581-588, 2006.
- [7] W.J. Hwang, Y.C. Chen and C.C. Hsu, "Robust transmission based on variable-rate error control and genetic programming," *IEEE Communication Letters*, Vol.6, pp.25-27, 2002.
- [8] W.J. Hwang, C.M. Ou, C.C. Hsu and T.Y. Lo, "Iterative optimization for joint design of source and channel codes using genetic algorithms," *Journal of the Chinese Institute of Engineers*, Vol. 28, pp.803-810, 2005.
- [9] P. Merz and B. Freisleben, "Genetic algorithms for binary quadratic programming," *Proc. the Genetic and Evolutionary Computation Conference*, pp.417-424, 1999.

- [10] H. Peng, F. Long, Z. Chi, W. Su, "A hierarchical distributed genetic algorithm for image segmentation," *Proc. of the 2000 Congress on Evolutionary Computation*, Vol. 1, pp. 272-276, July 2000.
- [11] D. Power, C. Ryna, R. M. A. Azad, "Promoting diversity using migration strategies in distributed genetic algorithms," *IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1831-1838, 2005.
- [12] N.J. Radcliffe and P.D. Surry, "Formal memetic algorithms," *Lecture Notes in Computer Science*, Vol. 865, pp.1-16, 1994.
- [13] M. Srinivas and L.M. Patnaik, "Genetic algorithm: a survey," *IEEE Computer*, Volume 27, pp. 17 - 26, 1994.
- [14] S. Tanaka, H. Kurata, T. Ohashi, "Optimization of E.Coli heat shock response parameter tuning using distributed and integrated genetic algorithms," *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, pp.1243-1248, 2004.
- [15] K. Zeger, J. Vaisey, and A. Gersho, "Globally optimal vector quantizer design by stochastic relaxation," *IEEE Trans. Signal Process.*, Vol. 40, pp. 310-322, 1992.

Chien-Min Ou received his diploma in Telecommunication Engineering from Chien Shin Institute of Technology, Chung Li, Taiwan, in 1978 and M.S., and Ph.D. degrees in Electrical Engineering from Chung Yuan Christian University, Chung Li, Taiwan, in 2000, and 2003, respectively. He joined the Faculty of the Department of Electronics Engineering, Ching Yun Institute of Technology, Chung Li, Taiwan, as a Lecturer in 1978. Since 2004, He has been an Assistant Professor of the Department of Electronics Engineering, Ching Yun University.

He is also a member of the honor society Phi Tau Phi. His research topics include VLSI design and testing, image processing, video compression, motion estimation, and genetic algorithm.

Jing-Jhih Chen was born in Kaohsiung, Taiwan, R.O.C., on November, 8, 1982. He received the B.S. degree in Electronics Engineering from Ching Yun University in 2005. He is presently working toward the M.S. degree in Electronics Engineering at the same university. His research interests include VLSI chip design and multimedia communications.

Wen-Jyi Hwang received his diploma in electronics engineering from National Taipei Institute of Technology, Taiwan, in 1987, and M.S.E.C.E. and Ph.D. degrees from the University of Massachusetts at Amherst in 1990 and 1993, respectively.

From September 1993 until January 2003, he was with the Department of Electrical Engineering, Chung Yuan Christian University, Taiwan. In February 2003, he joined the Graduate Institute of Computer Science and Information Engineering, National Taiwan Normal University, where he is now a Full Professor. His research interests are centered on multimedia communications systems with particular emphasis on image/video transmission.

Dr. Hwang is the recipient of the 2000 Outstanding Research Professor Award from Chung Yuan Christian University, 2002 Outstanding Young Researcher Award from the Asia-Pacific Board of the IEEE Communication Society, and 2002 Outstanding Young Electrical Engineer Award from the Chinese Institute of the Electrical Engineering.