Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2006) 130–138
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Concurrent genetic optimization for joint design of source and channel codes ☆

Chien-Min Ou[a], Wen-Jyi Hwang[b,*], Wen-Wei Hu[c], Tsung-Yan Lo[b]

[a]*Department of Electronics Engineering, Ching Yun University, Chungli, Taiwan 320, Taiwan ROC*
[b]*Graduate Institute of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan 117, Taiwan ROC*
[c]*Research and Design Division, Quanta Display Inc., Lungtan, Taiwan 325, Taiwan ROC*

Received 1 November 2004; received in revised form 27 February 2006; accepted 24 April 2006
Communicated by K.Li
Available online 8 July 2006

Abstract

A novel algorithm for jointly optimizing source and channel codes is presented in this paper. The algorithm uses the channel-optimized vector quantization (COVQ) for the source code, and rate-punctured convolutional coding (RCPC) for the channel code. The genetic algorithm (GA) is used for the concurrent design of both source and channel codes. The GA enhances the robustness of the rate–distortion performance of the COVQ to the selection of initial codewords. In addition, it reduces the computational time for realizing the unequal error protection scheme best matched to the COVQ. Numerical results show that the algorithm attains near optimal performance while having low computational complexity.

Keywords: Vector quantization; Genetic algorithm; Error correct coding

1. Introduction

The goal of the joint design of source and channel codes is to minimize the end-to-end average distortion of communication systems over a noisy channel. The basic techniques for the joint 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 optimally matched to a given noisy channel. Typical examples are the source-optimized vector quantization (COVQ) [4] 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 convolutional codes (RCPCs) [7] have been found to be effective for the implementation of the UEP. The application of RCPCs to 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 by Goldsmith and Effros [6] (thereby termed Goldsmith–Effros algorithm). 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 therefore is higher than the algorithms which only designs source or channel codes. In addition, the full-search bit allocation

☆This paper was presented in part at the Ninth European Conference on Genetic Programming (EuroGP 2006), Budapest, HUNGARY, April 2006.

*Corresponding author. Tel.: +886 2 2932 2421x201; fax: +886 2 2932 2378.
E-mail address: whwang@csie.ntnu.edu.tw (W.-J. Hwang).

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2006.04.008

scheme is usually used for realizing the UEP at each iteration in the scheme. The computational complexity of the full-search scheme may be very high for the long binary channel indices [8].

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 [17]. 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 [5], 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) [15]. 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 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 string 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 [11] and VQ design [9].

The objective of this paper is to present a new algorithm for the joint design of source and channel codes attaining superior rate–distortion performance using the hybrid GA. The Goldsmith–Effros algorithm [6], which involves a local search for source codes followed by a full search for channel codes, can be adopted as an effective tool for the local improvement to pure GA for the joint design. The resulting hybrid GA algorithm, however, may have high computational complexity because the channel codes are still designed by exhaustive search. It has been observed that the GA-based search for channel codes attains comparable performance to that of the exhaustive search [8]. To lower the computational time, the GA is also adopted for design of channel codes in [10]. Consequently, two pure GAs are used iteratively for joint design in [10] with one for source codes and the other for channel codes. Nevertheless, the cost function for the two GAs are different. The average distortion is used as the cost for the GA-based search of source codes; whereas, the transmission rate is part of the cost for channel codes. The rate–distortion performance of the design may be improved further if both the average distortion and transmission rate are considered concurrently for the search.

In light of the facts stated above, we employ a new hybrid GA technique using concurrent optimization for the joint design. In the new technique, the pure GA is used for the concurrent search of source and channel codes. Although the basic GA is adopted here for the sake of simplicity, more advanced GAs such as island GAs [2,16] and parallel GAs [3] can also be used to enhance the performance of the coarse search. 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 [8]. 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 remainder of this paper is stated as follows. In Section 2, we formulate the joint design problem. Section 3 then presents the GA-based approaches for solving the problem. The concurrent design algorithm will be described in detail in this section. The numerical results of the GA-based algorithms are then presented and compared in Section 4. Finally, Section 5 contains some concluding remarks of our work.

2. Problem formulation

Consider a full-search VQ with N codewords $\mathbf{y}_1, \ldots, \mathbf{y}_N$. Each codeword $\mathbf{y}_i$ is represented by a binary index c_i with length n, where $n = \log N$. Let $c_i(m)$, $m = 1, \ldots, n$, be the mth 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 1 in [7] (denoted by $\mathscr{C}$) are used to obtain our RCPC candidates. We represent each candidate code by a vector with dimension n, where the mth element in the vector is the channel code rate applied to $c_i(m)$. For example, suppose $n = 3$. The RCPC code $\{\frac{1}{2}, \frac{2}{3}, \frac{2}{3}\}$ applies the convolutional code with rate $\frac{1}{2}$ to $c_i(1)$, and the convolutional code with rate $\frac{2}{3}$ 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 $\{s_1, \ldots, s_n\}$, the average transmission rate is given by $\sum_{m=1}^{n} (1/s_m)$.

Table 1
Summary of the four GA-based algorithms for the joint design of source and channel codes

Algorithm	Target to optimize		Implementation
	Source code	Channel code	
G-COVQ	Yes	No	Memetic algorithm Coarse search: pure GA Local improvement: COVQ
G-UEP	No	Yes	Pure GA
GA-based iterative	Yes	Yes	Use G-COVQ and G-UEP iteratively
GA-based concurrent	Yes	Yes	Memetic algorithm Coarse search: pure GA for source and channel codes Local improvement: COVQ

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 $P_{k/i}$, $i,k=1,\ldots,N$, the index crossover probabilities, which are functions of RCPC and the BER of the BSC. Given source codewords and index crossover probabilities, the average end-to-end distortion, D, is given by

$$D=\frac{1}{wt}\sum_{j=1}^{t}\sum_{k=1}^{N}P_{k/\alpha(\mathbf{x}_j)}d(\mathbf{x}_j,\mathbf{y}_k), \tag{1}$$

where w 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 optimal source encoder minimizing D is then given by $\alpha(\mathbf{x}_j)=\arg\min_{1\leqslant i\leqslant N}\sum_{k=1}^{N}P_{k/i}d(\mathbf{x}_j,\mathbf{y}_k)$.

Given $\{\mathbf{x}_j\}_{j=1}^{t}$ and $\mathscr{C}$, the goal of joint design of source and channel codes is then defined by

$$\min_{\substack{(\mathbf{y}_1,\ldots,\mathbf{y}_N)\\(s_1,\ldots,s_n)}}\ \frac{1}{wt}\sum_{j=1}^{t}\sum_{k=1}^{N}P_{k/\alpha(\mathbf{x}_j)}d(\mathbf{x}_j,\mathbf{y}_k),$$
$$\text{s.t.}\quad\sum_{m=1}^{n}\frac{1}{s_m}\leqslant R, \tag{2}$$

where R is the constraint on the average transmission rate, and can be prespecified before the design.

To use the Lagrangian method for solving this optimization problem, the cost function to be minimized can be rewritten as

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

where the Lagrange multiplier λ determines the resulting transmission rate after the optimization [14]. Two extreme cases may deserve attention: $\lambda=0$ and ∞. When $\lambda=0$, the R attains the maximum value. That is, the mother code in $\mathscr{C}$ is used for the error protection of all bits in c_i. When $\lambda=\infty$, only the lowest degree of error protection is necessary, and the R is at its minimum.

3. The algorithms

In this section, we present four different joint design algorithms for the minimization of the cost function J given in Eq. (3): 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, and the GA-based concurrent design algorithm. Summary of these algorithms is given in Table 1.

3.1. G-COVQ algorithm

We first introduce the COVQ algorithm, which is the basic channel-optimized source coding technique for the joint design. In the COVQ design, we assume that the BER ε of the BSC channel and RCPC rates $\{s_1,\ldots,s_n\}$ are fixed. The index crossover probabilities $P_{k/i}$, $i,k=1,\ldots,N$, 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 $\mathbf{y}_k$, $k=1,\ldots,N$ minimizing D.

It can be shown that, given codewords $\mathbf{y}_k$, $k=1,\ldots,N$, the optimal source encoder α minimizing D should satisfy

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

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

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

The COVQ algorithm is based on an iterative procedure where source encoder α and codewords $\mathbf{y}_k$, $k=1,\ldots,N$ 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. One way to solve the problem is to combine the COVQ with GA, which is termed the G-COVQ algorithm.

Recall that, to employ the GA technique for solving an optimization problem, a number of genetic strings are first initialized and a fitness function for the regeneration of genetic strings is selected. After that, through the operations including the regeneration, mutation, and crossover a near global optimal solution to the optimization problem is obtained.

Suppose there are G strings in the G-COVQ algorithm. Each string $\mathbf{g}=\{\mathbf{y}_1,\ldots,\mathbf{y}_N\}_{\mathbf{g}}$ is a set of VQ codewords. Let $\mathscr{G}(q)$ be the set of G strings after the execution of the qth 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 $\mathscr{G}(0)$ 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 qth evolution is to be done. We then perform the following operations sequentially on the strings in $\mathscr{G}(q-1)$.

3.1.1. Regeneration of G-COVQ

Since each string in $\mathscr{G}(q-1)$ 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; that is, for the offspring generation, we spin a simulated biased roulette wheel whose slots have different sizes proportional to the fitness values of the individual strings. The results of spin gives a reproduction candidate.

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. There are G regeneration strings created after the regeneration operation.

3.1.2. Crossover of G-COVQ

On each regeneration string $\mathbf{g}$, $\{\mathbf{y}_1,\ldots,\mathbf{y}_N\}_{\mathbf{g}}$, the crossover operation is applied with probability P_c. Out of the total population, a partner string $\mathbf{g}'$, $\{\mathbf{y}'_1,\ldots,\mathbf{y}'_N\}_{\mathbf{g}'}$, is randomly chosen. Then an integer random numbers b (between 1 and N) is generated. The VQ codewords of both strings are cut into two portions at positions b. Portions of each segment of strings $\mathbf{g}$ and $\mathbf{g}'$ are mutually exchanged by the following way:

$$\{\mathbf{y}_1,\ldots,\mathbf{y}_b,\mathbf{y}_{b+1},\ldots,\mathbf{y}_N\}_{\mathbf{g}} \to \{\mathbf{y}_1,\ldots,\mathbf{y}_b,\mathbf{y}'_{b+1},\ldots,\mathbf{y}'_N\}_{\mathbf{g}},$$
$$\{\mathbf{y}'_1,\ldots,\mathbf{y}'_b,\mathbf{y}'_{b+1},\ldots,\mathbf{y}'_N\}_{\mathbf{g}'} \to \{\mathbf{y}'_1,\ldots,\mathbf{y}'_b,\mathbf{y}_{b+1},\ldots,\mathbf{y}_N\}_{\mathbf{g}'}.$$

3.1.3. Mutation of G-COVQ

For each string $\mathbf{g}$, mutation is performed on each element of $\mathbf{y}_k$, $k=1,\ldots,N$, 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.

3.1.4. Local improvement of G-COVQ

After the regeneration, crossover and mutation operations, the COVQ algorithm is applied to each string $\mathbf{g}$ for the local improvement. 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 $\mathscr{G}(q)$.

3.1.5. Optimal string updating and test for convergence of G-COVQ

After the completion of the COVQ optimization, the D value of each string in $\mathscr{G}(q)$ is computed. Let $\mathbf{g}^{**}$ be the string in $\mathscr{G}(q)$ 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 qth evolution of our genetic programming algorithm.

In the algorithm, the evolution continues until the observation of I consecutive evolutions yielding identical D^* value. To show the convergence of the algorithm, we first note that the D^* value after the completion of each evolution is either reduced or left unchanged. In addition, from Eq. (1), it is observed that the D^* value after each evolution is bounded below by zero. Hence, as the evolution continues, the sequence of D^* values is guaranteed to converge. The *current optimal* string $\mathbf{g}^*$ after the completion of GA algorithm is then chosen as the desired VQ codewords.

3.2. G-UEP algorithm

The UEP is the basic scheme for source-optimized channel coding, which can be realized using the full-search bit allocation. Suppose the number of RCPC code rates in $\mathscr{C}$ is K. Since the length of the binary index c_i representing codeword $\mathbf{y}_i$ is n, there are K^n possible allocations of RCPC if the full-search bit allocation scheme is used. Consequently, as n becomes large, the computational complexity of exhaustive search can be very high. One way to solve this problem is to use the GA for bit allocation. The algorithm, termed the GA-UEP algorithm, attains comparable performance to that of full-search with significantly lower computational complexity [8]. In the following, we give a brief description of the algorithm. Detailed description can be found in [8].

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

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

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

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

The problem of the UEP therefore is equivalent to finding a set of RCPC rates $\{s_1,\ldots,s_n\}$ minimizing the cost

function L given in Eq. (7) for a fixed set of VQ codewords. Suppose there are G strings. Each string $\mathbf{s} = \{s_1, \ldots, s_n\}_{\mathbf{s}}$ is a set of RCPC rates. Let $\mathscr{S}(q)$ be the set of G strings after the execution of the qth 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 $\mathscr{S}(0)$ are formed by randomly selecting channel rates in $\mathscr{C}$. Now, suppose the $(q-1)$th evolution is completed, and the execution of the qth iteration is to be done. We then perform the following genetic operations sequentially on the strings in $\mathscr{S}(q-1)$.

3.2.1. Regeneration of G-UEP

Each string in $\mathscr{S}(q-1)$ 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.

3.2.2. Crossover of G-UEP

On each regeneration string $\mathbf{s}$, $\{s_1, \ldots, s_n\}_{\mathbf{s}}$, one point crossover is applied with probability P_c. Out of the total population, a partner string $\mathbf{s}'$, $\{s'_1, \ldots, s'_n\}_{\mathbf{s}'}$, 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 $\{s_{b+1}, \ldots, s_n\}$ and $\{s'_{b+1}, \ldots, s'_n\}$ are mutually exchanged.

3.2.3. Mutation of G-UEP

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

3.2.4. Optimal string updating and test for convergence of G-UEP

The G strings after these operations are then the strings of the set $\mathscr{S}(q)$. The L value of each string in $\mathscr{S}(q)$ is computed. Based on these L values, the $\mathbf{s}^*$ and L^* can be updated in the way similar to that for updating $\mathbf{g}^*$ and D^* in the G-COVQ. This completes the execution of qth evolution of G-UEP algorithm.

In the G-UEP algorithm, the evolution continues until the observation of I consecutive evolutions yielding identical L^* value. The convergence of L^* value can be proved in the similar manner to that of D^* in the G-COVQ. The *current optimal* string $\mathbf{s}^*$ after the completion of G-UEP algorithm is then chosen as the desired RCPC rates.

3.3. GA-based iterative algorithm

The iterative combination of the G-COVQ and G-UEP can further improve the performance of the joint design. In the iterative algorithm, each iteration executes the G-COVQ and G-UEP sequentially. Let $\{\mathbf{y}_1^f, \ldots, \mathbf{y}_N^f\}$ and $\{s_1^f, \ldots, s_n^f\}$ be the set of VQ codewords and RCPC rates after the design of the fth iteration, respectively. Now, suppose the $(f-1)$th iteration is completed, and the design of the fth iteration is to be done. Each iteration contains two steps, which correspond to G-COVQ and G-UEP design, respectively.

3.3.1. Step 1

Given $\{s_1^{f-1}, \ldots, s_n^{f-1}\}$, the objective at this step is to design $\{\mathbf{y}_1^f, \ldots, \mathbf{y}_N^f\}$ using the G-COVQ algorithm with fitness function $1/D$. The set of RCPC rates $\{s_1^{f-1}, \ldots, s_n^{f-1}\}$ is used to determine the index crossover probabilities $P_{k/i}$, $i, k = 1, \ldots, N$, for the computation of D given in Eq. (1). The VQ codewords $\{\mathbf{y}_1^f, \ldots, \mathbf{y}_N^f\}$ at iteration f is then set to be the final current optimal string $\mathbf{g}^*$ after the completion of G-COVQ.

3.3.2. Step 2

Using the G-UEP with fitness function $1/L$, this step finds the RCPC rates $\{s_1^f, \ldots, s_n^f\}$ best matched to the VQ codewords $\{\mathbf{y}_1^f, \ldots, \mathbf{y}_N^f\}$ designed at the previous step. The VQ codewords are used to compute the first term in L shown in Eq. (7) (i.e., $(1/N)\sum_{i=1}^{N}\sum_{k=1}^{N} P_{k/i} d(\mathbf{y}_i, \mathbf{y}_k)$) for the execution of the G-UEP. The RCPC rates $\{s_1^f, \ldots, s_n^f\}$ at the iteration f is then set to be the final current optimal string $\mathbf{s}^*$ after the completion of G-UEP.

3.3.3. Test for convergence of the iterative algorithm

Note that, in the G-COVQ and G-UEP, the RCPC rates and VQ codewords are fixed, respectively. Therefore, minimizing D and L is equivalent to minimizing J in the G-COVQ and G-UEP, respectively. Let J^f be the value of J after the completion of the fth iteration. Since each application of G-COVQ and G-UEP reduce or retain the J value, the sequence $\{J^f\}$ is non-increasing. In addition, the J^f is bounded below by zero. The convergence of sequence $\{J^f\}$ is guaranteed. The iterative algorithm will continue until the condition $(|J^{f-1} - J^f|/J^f) \leqslant \delta$ is satisfied, where $\delta > 0$ is a prespecified small positive number.

3.4. GA-based concurrent algorithm

In the GA-based concurrent algorithm, each string $\mathbf{g}$ in the algorithm can be divided into two segments: the VQ codewords segment $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}_{\mathbf{g}}$ and the RCPC rates segment $\{s_1, \ldots, s_n\}_{\mathbf{g}}$. Let $\mathscr{G}(q)$ be the set of G strings after the execution of the qth 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 and J^* be its J value. We set the initial $\mathbf{g}^*$ as null, and initial $J^* = \infty$. In addition, the VQ codewords and RCPC rates of each string in $\mathscr{G}(0)$ are formed by randomly selecting source vectors and channel rates in $\{\mathbf{x}_j\}_{j=1}^{t}$ and $\mathscr{C}$, respectively. Now, suppose the $(q-1)$th evolution is completed, and the execution of the qth evolution is to be

done. We then perform the following genetic operations sequentially on the strings in $\mathscr{G}(q-1)$.

3.4.1. Regeneration of GA-based concurrent algorithm

Each string in $\mathscr{G}(q-1)$ contains both VQ codewords and RCPC rates. Therefore, we use the inverse of J given in Eq. (3) as the fitness function for each string in $\mathscr{G}(q-1)$. 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.

3.4.2. Crossover of GA-based concurrent algorithm

On each regeneration string $\mathbf{g}$, $\{\mathbf{y}_1,\ldots,\mathbf{y}_N,s_1,\ldots,s_n\}_{\mathbf{g}}$, the crossover operation is applied with probability P_c. Out of the total population, a partner string $\mathbf{g}'$, $\{\mathbf{y}'_1,\ldots,\mathbf{y}'_N, s'_1,\ldots,s'_n\}_{\mathbf{g}'}$, 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 $\mathbf{g}$ and $\mathbf{g}'$ are mutually exchanged by the following way:

$$\{\mathbf{y}_1,\ldots,\mathbf{y}_{b_1},\mathbf{y}_{b_1+1},\ldots,\mathbf{y}_N\}_{\mathbf{g}} \rightarrow \{\mathbf{y}_1,\ldots,\mathbf{y}_{b_1},\mathbf{y}'_{b_1+1},\ldots,\mathbf{y}'_N\}_{\mathbf{g}},$$
$$\{s_1,\ldots,s_{b_2},s_{b_2+1},\ldots,s_n\}_{\mathbf{g}} \rightarrow \{s_1,\ldots,s_{b_2},s'_{b_2+1},\ldots,s'_n\}_{\mathbf{g}},$$
$$\{\mathbf{y}'_1,\ldots,\mathbf{y}'_{b_1},\mathbf{y}'_{b_1+1},\ldots,\mathbf{y}'_N\}_{\mathbf{g}'} \rightarrow \{\mathbf{y}'_1,\ldots,\mathbf{y}'_{b_1},\mathbf{y}_{b_1+1},\ldots,\mathbf{y}_N\}_{\mathbf{g}'},$$
$$\{s'_1,\ldots,s'_{b_2},s'_{b_2+1},\ldots,s'_n\}_{\mathbf{g}'} \rightarrow \{s'_1,\ldots,s'_{b_2},s_{b_2+1},\ldots,s_n\}_{\mathbf{g}'}.$$

3.4.3. Mutation of GA-based concurrent algorithm

For each string $\mathbf{g}$, mutation is performed on each element of $\mathbf{y}_k$, $k=1,\ldots,N$, and s_m, $m=1,\ldots,n$, 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. For each s_q determined to be mutated, a rate is first selected at random from $\mathscr{C}$, and s_q is then replaced by the rate.

3.4.4. Local improvement of GA-based concurrent algorithm

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 $\mathscr{G}(q)$.

3.4.5. Optimal string updating and test for convergence of GA-based concurrent algorithm

After the completion of the COVQ optimization, the J value of each string in $\mathscr{G}(q)$ is computed. The new $\mathbf{g}^*$ and J^* can then be obtained in the way similar to that for updating $\mathbf{g}^*$ and D^* in the G-COVQ. This completes the execution of qth evolution of GA-based concurrent algorithm.

In the algorithm, the evolution will continue until the observation of I consecutive evolutions yielding identical J^* value. The proof of the convergence of J^* values follows the procedure similar to that of D^* in the G-COVQ. The *current optimal* string $\mathbf{g}^*$ after the completion of GA algorithm is then chosen as the desired VQ codewords and RCPC rates for the robust transmission.

4. 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 Goldsmith–Effros algorithm [6] 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 84 000.

Fig. 1 elaborates the dependence of the GA-based concurrent algorithm on the population size G. In this experiment, the number of codewords is $N=32$. Accordingly, the length of binary index representing each codeword is $n=5$. Since the GA is a stochastic optimization process, each application of the GA may have different results. Therefore, each sample point in the figure is an average value over 100 independent executions using

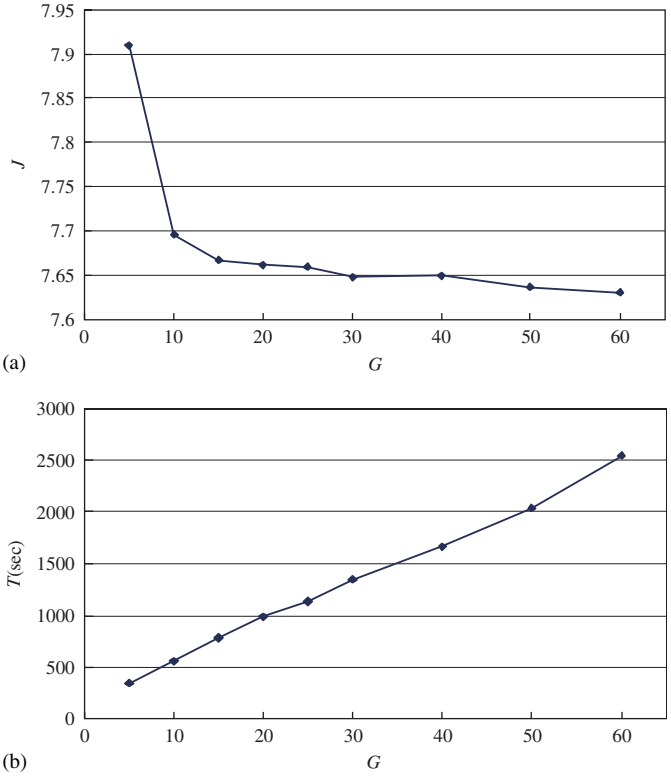

Fig. 1. The dependence of the GA-based concurrent algorithm on the population size G. (a) Average cost J verus G, (b) average CPU time T verus G, where the CPU time is measured on 1.6 GHz Pentium IV CPU.

Table 2
The average cost J and CPU time T (measured on 1.6 GHZ Pentium IV) of the GA-based concurrent algorithm with $\lambda = 1$ and $G = 15$ for different pairs of (P_c, P_m) parameters

	$P_c = 0.2$		$P_c = 0.4$		$P_c = 0.6$		$P_c = 0.8$	
	J	T (min)	J	T (min)	J	T (min)	J	T (min)
$P_m = 0.2$	7.71	10.53	7.69	9.94	7.69	13.01	7.70	9.62
$P_m = 0.4$	7.72	10.17	7.71	9.53	7.73	12.19	7.72	13.17
$P_m = 0.6$	7.73	10.10	7.73	9.47	7.73	12.28	7.73	10.30
$P_m = 0.8$	7.75	10.06	7.76	12.21	7.77	12.14	7.75	9.52

Table 3
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
20	114.77	128.94	115.19	1.47	118.77	139.07	126.15	7.15

randomly chosen initial genetic strings. Fig. 1(a) shows the resulting average cost (i.e., J in Eq. (3)) obtained by the GA-concurrent algorithm with different G values, where we set $\lambda = 1.0$ for the optimization. From the figure, we observe that the average J decreases as G increases. However, the reduction becomes negligible when $G \geqslant 15$. Moreover, from Fig. 1(b), it follows that the computational complexity, which is defined as the average CPU time required for each execution of the algorithm, grows linearly with G. Therefore, we choose $G = 15$ for the subsequent experiments so that both low computational complexity and high performance can be attained.

Table 2 shows the average cost J and computational complexity of the GA-based concurrent algorithm with $\lambda = 1$ and $G = 15$ for different pairs of (P_c, P_m) parameters. Both P_c and P_m take values from the set $\{0.2, 0.4, 0.6, 0.8\}$. Therefore, there are 16 pairs of parameters considered in Table 2. We conclude from the table that the average cost J and computational complexity of the GA-based concurrent algorithm are insensitive to a wide range of variations on parameters.

Table 3 compares the GA-based concurrent algorithm with its major counterpart, the GA-based iterative algorithm for $P_c = 0.6$ and $P_m = 0.2$. 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

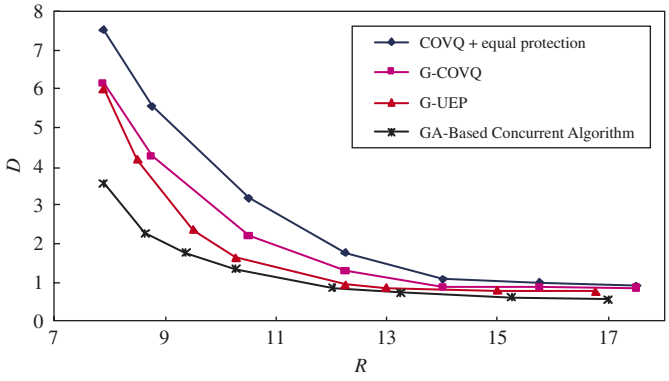

Fig. 2. The rate–distortion performance of COVQ, G-COVQ, G-UEP and GA-based concurrent algorithms.

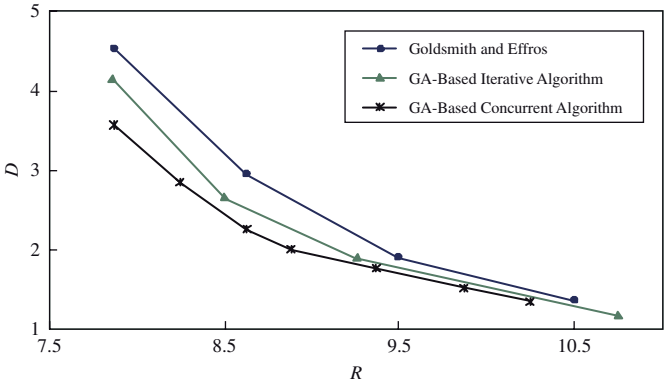

Fig. 3. The rate–distortion performance of Goldsmith and Effros, GA-based iterative and GA-based concurrent algorithms.

Table 4
The average CPU time (measured on 1.6 GHz Pentium IV) of various algorithms for the experiments shown in Figs. 2 and 3

Algorithm	G-COVQ	GA-based iterative	GA-based concurrent	Goldsmith and Effros
CPU time	1.9 h	7.3 h	2.1 h	60.3 h

provides robust solutions with superior performance. The GA-base 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 after the convergence of the algorithm.

In addition to evaluating the cost J of the GA-based concurrent algorithm for the joint design problem, we also compare the rate–distortion performance of the algorithm with that of other techniques, as shown in Figs. 2 and 3. The average distortion D in the figures is defined as the mean squared distance between the original and reconstructed test Gauss–Markov sequences. The average transmission rate is defined in Eq. (6). To achieve meaningful comparisons, all the algorithms have the same number of VQ codewords $N = 128$.

It is not surprising to observe from Fig. 2 that the GA-based concurrent algorithm significantly outperforms the COVQ, G-COVQ and G-UEP algorithms, which are only the channel-optimized source coding technique or the source-optimized channel coding technique, and do not optimize both the source and channel codes. In addition, as illustrated in Fig. 3, the GA-based concurrent algorithm also has superior rate–distortion performance over the GA-based iterative algorithm and Goldsmith–Effros algorithm [6], 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 3, where the concurrent algorithm attains lower cost for the optimization.

The Goldsmith–Effros algorithm is an iterative algorithm without the employment of 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 4 shows the computational complexity of various algorithms for the experiments shown in Figs. 2 and 3. 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 h, which is significantly lower than that of the Goldsmith–Effros algorithm (60.3 h). 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.

5. Conclusion

The concurrent optimization algorithm using the hybrid GA has been found to be effective for the joint design of source and channel codes. It achieves a near optimal performance while maintaining lower computational complexity. In our experiments, the algorithm attains superior rate–distortion performance with lower computational complexity as compared to its iterative counterparts. In addition, its performance is robust to the selection of parameters. The algorithm therefore can be an effective alternative for the applications where both low computational complexity and high performance are desired for the design of robust transmission systems.

References

[1] S. Areibi, M. Moussa, H. Abdullah, A comparison of genetic/memetic algorithms and heuristic searching, Proceedings of the International Conference on Artificial Intelligence, June 2001, pp. 660–666.
[2] T.C. Belding, The distributed genetic algorithms revisited, Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 114–121.
[3] E. Cantu-Paz, A survey of parallel genetic algorithms, Technical Report 97003, Department of Computer Science and Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, 1997.
[4] N. Farvardin, V. Vaishampayan, On the performance and complexity of channel-optimized vector quantizers, IEEE Trans. Inform. Theory 37 (1991) 155–160.
[5] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. 6 (1984) 721–741.
[6] A.J. Goldsmith, M. Effros, Joint design of fixed-rate source codes and multiresolution channel codes, IEEE Trans. Commun. 46 (1998) 1301–1311.
[7] J. Hagenauer, Rate-compatible punctured convolutional codes (RCPC) codes and their applications, IEEE Trans. Commun. 36 (1988) 389–400.
[8] W.J. Hwang, Y.C. Chen, C.C. Hsu, Robust transmission based on variable-rate error control and genetic programming, IEEE Commun. Lett. 6 (2002) 25–27.
[9] W.J. Hwang, S.L. Hong, Genetic entropy-constrained vector quantization, Opt. Eng. 38 (1999) 233–239.
[10] W.J. Hwang, C.M. Ou, C.C. Hsu, T.Y. Lo, Iterative optimization for joint design of source and channel codes using genetic algorithms, J. Chinese Inst. Eng. 28 (2005) 803–810.
[11] P. Merz, B. Freisleben, Genetic algorithms for binary quadratic programming, Proceedings of the Genetic and Evolutionary Computation Conference, 1999, pp. 417–424.
[12] N.J. Radcliffe, P.D. Surry, Formal Memetic Algorithms, Lecture Notes in Computer Science, vol. 865, 1994, pp. 1–16.
[14] Y. Shoham, A. Gersho, Efficient bit allocation for an arbitrary set of quantizers, IEEE Trans. Acoustics, Speech Signal Process. 36 (1988) 1445–1453.

[15] M. Srinivas, L.M. Patnaik, Genetic algorithm: a survey, IEEE Comput. 27 (1994) 17–26.
[16] R. Tanese, Distributed genetic algorithms, Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 432–439.
[17] K. Zeger, J. Vaisey, A. Gersho, Globally optimal vector quantizer design by stochastic relaxation, IEEE Trans. Signal Process. 40 (1992) 310–322.

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, 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.

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.

Wen-Wei Hu was born in I-Lan, Taiwan, R.O.C., on September 27, 1979. He received the B.S., and M.S. degrees in Electrical Engineering from Chung Yuan Christian University, Chung Li, Taiwan, in 2002, and 2004, respectively. From October 2004, he joined the Quanta Display Inc. as a research engineer. His research interests include image coding and digital signal processing.