AFRL-SR-AR-TR-03-

0297

REPORT DOCUMENTATION PAGE

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
15 July 2003	Final Technical Report	1 Jan. 2000-30 Nov. 2002

4. TITLE AND SUBTITLE
Robust Mobile Multimedia Communications

5a. CONTRACT NUMBER

5b. GRANT NUMBER
F49620-00-1-0117

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Lance C. Pérez
Michael W. Hoffman
Khalid Sayood

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical Engineering
209N WSEC
Univ. of Nebraska, Lincoln
Lincoln, NE 68588-0511

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of
Scientific Resesearch (AFOSR)

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
N/A

Approved for public release,
distribution unlimited

20030822 144

13. SUPPLEMENTARY NOTES
N/A

14. ABSTRACT

This report describes the final results obtained on this project. We detail the technical contributions made under the support of this grant. The technical contributions are in the two focus areas

1. Time-varying convolutional codes and their application to turbo codes.
2. Joint-source channel coding for multimedia applications.

Results in the area of time-varying convolutional codes are discussed including the discovery of a new time-varying convolutional code that achieves the Heller bound and has free distance greater than the best time-variant convolutional code. The report also investigates the application of time-varying convolutional codes as component codes in turbo codes. The work done in this area resulted in a new time-varying component code that outperforms the celebrated big numerator, little denominator (BNLD) codes of Massey, Takeshita and Costello. Finally, this report contains a thorough simulation comparison of the state of the art of joint-source channel coding schemes for image transmission.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

Robust Mobile Multimedia Communications

Final Technical Report for
Grant Number F49620-00-1-0117
1 January 2000 to 30 November 2002

submitted to
Dr. Jon Sjogren
Director, Signals Communication and Surveillance
Air Force Office of Scientific Research

Lance C. Pérez, Michael W. Hoffman and Khalid Sayood
with contributions from
Christopher G. Hruby, Fan Jiang and Devrim Ayilldiz

Department of Electrical Engineering
University of Nebraska, Lincoln
Lincoln, NE 68588-0511

15 July 2003

Contents

List of Figures

List of Tables

1 Summary

This grant supported research in the area of robust mobile multimedia communications with two focus areas:

- Time-varying convolutional codes and their application to turbo codes.
- Joint-source channel coding for multimedia communications.

The work performed under this grant led to significant contributions in both areas and resulted in three (3) conference papers and three (3) journal papers. A detailed discussion of the technical results forms the majority of this report and begins in Section 2.

The grant and matching funds provided partial support for three (3) faculty members and six (6) graduate students, including two U.S. citizens. The work performed by these students has resulted in three (3) Master's degrees and will result in an additional two (2) Master's degree and one (1) Doctoral degree. In addition, worked performed under this grant is included in the book *Trellis and Turbo Coding* by Christian B. Schlegel, of the University of Alberta, and Lance C. Pérez, of the University of Nebraska, Lincoln, to be published by the IEEE Press/John Wiley and Sons in the Fall of 2003.

1.1 Personnel Supported

Principle Investigator: Lance C. Pérez, Ph.D.

Co-Principle Investigator: Michael W. Hoffman, Ph.D.

Co-Principle Investigator: Khalid Sayood, Ph.D.

Research Assistant: Qian Hu, M.S.E.E., University of Nebraska, Lincoln, December 2000.

Research Assistant: Gulay Ozkan, M.S.E.E., University of Nebraska, Lincoln, December 2000.

Research Assistant: Devrim Ayilldiz, M.S.E.E.,University of Nebraska, Lincoln, May 2001.

Research Assistant: Christopher G. Hruby, M.S.E.E. University of Nebraska, Lincoln, expected August 2003.

Research Assistant: Matthew Becker, M.S.E.E. University of Nebraska, Lincoln, expected August 2003.

Research Assistant: Fan Jiang, Ph.D. University of Nebraska, Lincoln, expected August 2004

1.2 Technical Publications

Journal Publications

1. Q. Bao, M. W. Hoffman, and K. Sayood, "Multiple description imaged coding by vector quantization", submitted to the IEEE Transactions on Circuits and Systems for Video Technology.
2. Q. Hu and L. C. Pérez, "Time-varying convolutional codes that meet the Heller bound", in preparation for the IEEE Transactions on Information Theory.
3. F. Jiang, M. Becker, and L. C. Pérez, "Time-varying turbo codes", in preparation for IEEE Communications Letters.

Reviewed Conference Proceedings

1. F. Jiang, M. Becker and L. C. Pérez, "Turbo Codes with Time-Varying Component Codes", 2003 Conference on Information Sciences and Systems, The Johns Hopkins University, Baltimore, MD, March 13, 2003.
2. Q. Hu and L. C. Pérez, "Some Periodic Time-Varying Convolutional Codes with Free Distance Achieving the Heller Bound,' 2001 International Symposium on Information Theory, Washington D.C., June 28, 2001.
3. Q. Hu and L. C. Pérez, "Time-Varying Convolutional Codes with Large Distance," 2001 Conference on Information Sciences and Systems, The Johns Hopkins University, Baltimore, MD, March 21, 2001.

Masters Theses

1. Q. Bao, *Multiple Description Image Coding Over Noise Channels by Vector Quantization*, May, 2000.
2. G. Ozkan, *Design of Variable Length Codes for Noisy Channels*, December 2000.
3. Q. Hu, *Design and Analysis of Time-Varying Convolutional Codes*, December 2000.
4. M. Becker, *Practical Issues of Turbo Coding and Iterative Decoding*, expected August 2003.
5. C. Hruby *Properties of Low Density Parity Check Codes*, expected August 2003.

Doctoral Dissertations

1. F. Jiang, *Structural Properties and Applications of Time-varying Convolutional Codes*, expected August 2004.

2 Technical Results

In the remainder of this report, we detail the technical contributions made under the support of this grant. The technical contributions are in the two focus areas

- Time-varying convolutional codes and their application to turbo codes.
- Joint-source channel coding for multimedia applications.

Section 3 discusses results in the area of time-varying convolutional codes. This includes the discovery of a new time-varying convolutional code that achieves the Heller bound and has free distance greater than the best time-variant convolutional code. Section 4 discusses the application of time-varying convolutional codes as component codes in turbo codes. The work done in this area resulted in a new time-varying component code that outperforms the celebrated big numerator, little denominator (BNLD) codes of Massey, Takeshita and Costello [14]. Finally, section 5 contains a thorough simulation comparison of the state of the art of joint-source channel coding schemes for image transmission.

3 Time-Varying Convolutional Codes

3.1 Periodic Time Varying Convolutional Codes

In general, a periodic time varying convolutional code is denoted as a (n, k, m, P) code, where n, k, and m are defined in the same way as for time invariant codes and P is the period of the code. The ordered set $\{C_i, i = 1, 2, \ldots, P\}$ represents the time invariant codes that make up the periodic time varying code.

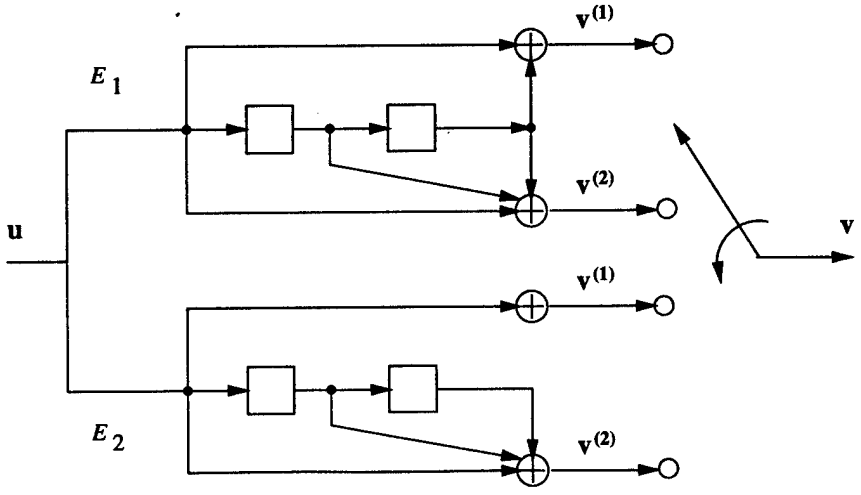

Figure 1: Encoder for a (2,1,2,2) periodic time varying convolutional code.

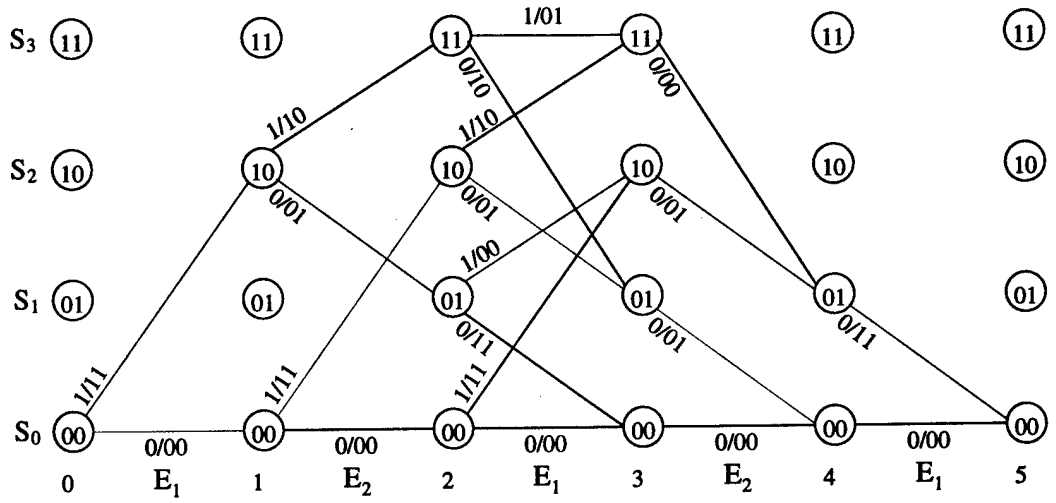

Figure 2: Trellis diagram for a (2,1,2,2) periodic time varying convolutional code.

Figure 1 shows a $(2, 1, 2, 2)$ periodic time varying code, where E_1 is the encoder of a $(2, 1, 2)$ time invariant code, C_1, with $g^{(1)} = (1\ 0\ 1)$ and $g^{(2)} = (1\ 1\ 1)$, and E_2 is the encoder of a code, C_2, with $g^{(1)} = (1\ 0\ 0)$ and $g^{(2)} = (1\ 1\ 1)$. The input, $\mathbf{u}$, enters both E_1 and E_2, while the output $\mathbf{v}$ switches between the outputs of E_1 and outputs of E_2. The state of E_1 and the state of E_2 are the same at all time. This can also be understood as E_1 and E_2 share the same memory. Therefore, E_1 and E_2 have the same transitions in the state diagram but with different branch labels.

The trellis of the periodic time varying code is obtained by expanding the state diagram in time and labeling branches alternatively with the branch label of E_1 and E_2. Figure 2 shows the trellis of the encoder in Figure 1. Let the input sequence $\mathbf{u}_{[0,\,6)} = (1\ 0\ 1\ 0\ 0)$. The encoding of this sequence by the periodic time varying code is shown in Table 1 and its trellis is the blue path shown in Figure 2. At time unit 0, "1" goes into E_1 and E_2, the state changes from S_0 to S_2 and output becomes (1 1) since E_1 is on duty. Then at time unit 1, "0" enters and the state transits to S_1. The output switches to E_2 and (0 1) is the output of the periodic time varying code at time unit 1. The encoding result is shown in Table 1. This can also be obtained by following the blue path in Figure 2.

Time	input $\mathbf{u}$	E_1 output	E_2 output
0	1	11	
1	10		01
2	101	00	
3	1010		01
4	10100	11	

Table 1: Encoding of the code in Figure 1 with input sequence $\mathbf{u}_{[0,\,6)} = (1\ 0\ 1\ 0\ 0)$.

From Chapter 2, $\mathbf{u}_{[i,\,j)}$ and $\mathbf{v}_{[i,\,j)}$ denote the input and output sequences, respectively, over the time instants $i,\ i+1,\ \cdots,\ j-1$ and $\mathbf{u}_t$ is the k-tuple of input bits at time instant t. The free distance for a linear time invariant convolutional code is the smallest Hamming distance between output sequences $\mathbf{v}_{[0,\,\infty)}$ resulting from distinct input sequences $\mathbf{u}_{[0,\,\infty)}$. It is equivalent to the

smallest $\mathbf{w}_H(\mathbf{v}_{[0,\,\infty)})$ achievable with $\mathbf{u}_{[0,\,\infty)} \neq 0$, where $\mathbf{w}_H(\cdot)$ denotes Hamming weight. For periodic time varying codes, d_{free} is equal to the smallest $\mathbf{w}(\mathbf{v}_{[0,\,\infty)})$ achievable with $\mathbf{u}_{[0,\,P)} \neq 0$. For time invariant codes, $\mathbf{u}_{[0,\,\infty)}$ may be considered to be all information sequences with $\mathbf{u}_0 \neq 0$. This corresponds to all the paths in the trellis diagram that leave S_0 at time 0.

The condition for periodic time varying codes, $\mathbf{u}_{[0,\,P)} \neq 0$, states that the information sequence can not be all zero from time 0 to $P-1$. These information sequences correspond to all the paths leaving S_0 at time 0, 1, ..., or $P-1$. For example, the free distance of the code in Figure 1 is 4. The path which has weight 4 is shown as the red path in Figure 2. This path corresponds to the information sequence $\mathbf{u}_{[0,\,4)} = (0\ 1\ 0\ 0)$. It is the only path of weight 4. This path leaves S_0 at time 1. Note that the first two information bits are (0 1). This satisfies the condition that $\mathbf{u}_{[0,\,P)} \neq 0$ with $P = 2$.

The column distance of a periodic time varying code is defined in almost the same way as it is in the time invariant case, except that instead of only one column distance $d^p_{c,i}$ at each time unit i, there are P column distances $d^p_{c,i}(j)$, $j = 1,\ 2,\ \ldots,\ P$, corresponding to the initial code C_j, $j = 1,\ 2,\ \ldots,\ P$ at time unit i. That is,

$$\begin{aligned} d^p_{c,i}(j) &= \min\{d(\mathbf{v}'_{[0,\,i)}, \mathbf{v}''_{[0,\,i)}) : \mathbf{u}'_0 \neq \mathbf{u}''_0 \quad C_j \text{ initial}\} \\ &= \min\{\mathbf{w}(\mathbf{v}_{[0,\,i)}) : \mathbf{u}_0 \neq \mathbf{0} \quad C_j \text{ initial}\} \end{aligned}$$

where $i = 0,\ 1,\ \ldots$ and $j = 1,\ \ldots,\ P$. It is easy to find that $d^p_{c,2}(1) = 3$ and $d^p_{c,2}(2) = 3$ for the code in Figure 1 through its trellis in Figure 2.

Like the column distance, there are P distance profiles for a periodic time varying code. They are defined as

$$\mathbf{d}^p(j) = [d^p_{c,0}(j),\ d^p_{c,1}(j),\ \cdots,\ d^p_{c,m}(j)] \qquad j = 1,\ 2,\ \ldots,\ P$$

where j corresponds to the initial code C_j. For instance, $\mathbf{d}^p(0)$ and $\mathbf{d}^p(1)$ for the periodic time varying code in Figure 1 are

$$\mathbf{d}^p(0) = [2,\ 3,\ 3]$$

and

$$\mathbf{d}^p(1) = [2,\ 3,\ 3].$$

It might happen that the time invariant MFD convolutional code and the periodic time varying convolutional code for a given rate and memory have the same free distance. For example, the $(2,1,2)$ time invariant code with $\mathbf{g} = [7,2]$ has $d_{free} = 4$ and the periodic time varying code with the encoder shown in Figure 1 also has $d_{free} = 4$. Thus, to resolve ties in d_{free}, two other criteria are introduced. The first criterion is $N_{d_{free}}$, the average number of paths of weight d_{free} per time instant. The second criterion is $I_{d_{free}}$, the average number of information bits per time instant along paths with weight d_{free}. If two codes have the same d_{free}, the one with the smaller value of $N_{d_{free}}$ and then $I_{d_{free}}$ is said to be better. Mathematically, $N_{d_{free}}$ and $I_{d_{free}}$ are defined as

$$N_{d_{free}} = \frac{1}{P}\#\{\mathbf{u}_{[0,\,\infty)} : \mathbf{u}_{[0,\,P)} \neq 0, \mathbf{w}_H(\mathbf{v}_{[0,\,\infty)}) = d_{free}\}$$

and

$$I_{d_{free}} = \frac{1}{P} \sum_{\mathbf{u}_{[0,\,\infty)} \in \mathbf{s}} \mathbf{w}_H(\mathbf{u}_{[0,\,\infty)}),$$

where $\#\{\cdot\}$ denotes cardinality and $\mathbf{s}$ is the set of input sequences which cause the output sequences with Hamming weight d_{free}.

The $N_{d_{free}}$ and $I_{d_{free}}$ of the code in Figure 1 are 1/2 and 1/2. There is only one path of weight 4 for this code. Therefore, $N_{d_{free}} = 1/2$. The corresponding input is $\mathbf{u}_{[0,\,4)} = (0\ 1\ 0\ 0)$. The weight of this information sequence is 1. Since there is only one path, the sum of the weight is 1 and $I_{d_{free}} = 1/2$.

3.2 Previous Results

Motivated by Costello's conjecture, early work on periodic time varying codes focus on searching for periodic time varying codes with larger d_{free} than the comparable time invariant codes. Mooser[1] found that some periodically time varying convolutional codes have the same d_{free} as the time invariant MFD code, but have both a smaller $N_{d_{free}}$ and $I_{d_{free}}$. The codes he found are $(2, 1, 4, P)$ periodic convolutional codes with $P = 1$, 2 and 3 and are listed in Table 2. Note that the time invariant code, [23, 35] is the MFD time invariant code. Simulation results for these codes are shown in Figure 3. Mooser concentrated on memory 4, rate 1/2 codes, because it is the smallest memory M such that Heller's upper bound on $d_{free} = 8$ is not achieved by any time invariant convolutional code. However, Mooser did not find a periodic time varying code with memory 4 and rate 1/2 that has d_{free} equal to 8.

P	r	d_{free}	$N_{d_{free}}$	$I_{d_{free}}$	Generator Sequences (Octal)
1	1/2	7	2	4	[23, 35]
2	1/2	7	3/2	3	[23, 35] [25, 37]
3	1/2	7	4/3	8/3	[23, 35] [25, 37] [27, 35]

Table 2: The $(2, 1, 4)$ time invariant MFD code and the $(2, 1, 4, P)$ periodic time varying convolutional codes found by Mooser with $P = 2$ and 3.

Palazzo[2] found some $(4, 1, 2, P)$ periodic time varying convolutional codes with $P = 1$, 2, 3, 4 and 5, which have the same d_{free} as the time invariant MFD codes, but smaller $N_{d_{free}}$ and $I_{d_{free}}$. These codes are listed in Table 3. Simulation results for these codes are shown in Figure 4. Note that the $[57^3]$ code is the MFD time invariant code. Here, 57^3 means the generator sequences, g_{input}^{output}, are: $g_0^{(0)} = 5$, $g_0^{(1)} = 7$, $g_0^{(2)} = 7$ and $g_0^{(3)} = 7$.

P	r	d_{free}	$N_{d_{free}}$	$I_{d_{free}}$	Generator Sequences (Octal)
1	1/4	10	1	2	57^3
2	1/4	10	1/2	1/2	$5^2 7^2\ 57^3$
3	1/4	10	2/3	2/3	$5^2 7^2\ 5^2 7^2\ 57^3$
4	1/4	10	3/4	3/4	$5^2 7^2\ 5^2 7^2\ 5^2 7^2\ 57^3$
5	1/4	10	4/5	4/5	$5^2 7^2\ 5^2 7^2\ 5^2 7^2\ 5^2 7^2\ 57^3$

Table 3: The $(4, 1, 2)$ time invariant MFD code and $(4, 1, 2, P)$ periodic time varying convolutional codes found by Palazzo with $P = 2$, 3, 4, and 5.

Palazzo[2] also found a $(3, 2, 1, 2)$ periodic time varying code with d_{free} greater than the time invariant MFD code. This code is shown in Table 4, where $[6, 4, 0\ ;\ 2, 6, 6]$ means g_{input}^{output} are:

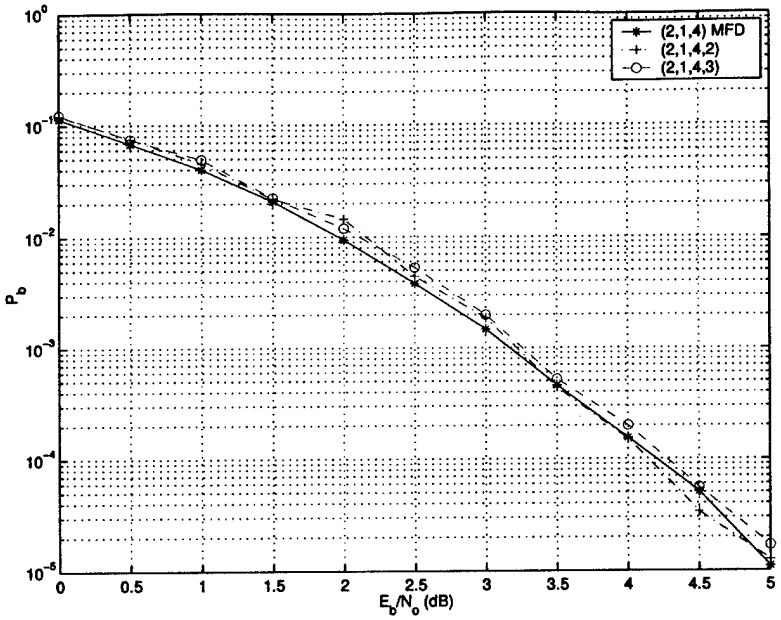

Figure 3: Simulation results of the time invariant MFD code and the $(2,1,4,P)$ codes found by Mooser.

$g_0^{(0)} = (1\ 1\ 0)$, $g_0^{(1)} = (1\ 0\ 0)$, $g_0^{(2)} = (1\ 1\ 0)$, $g_1^{(0)} = (0\ 1\ 0)$, $g_1^{(1)} = (1\ 1\ 0)$ and $g_1^{(2)} = (1\ 1\ 0)$. The $(3,2,1)$ MFD time invariant code is listed in Table 4, too. The simulation results of these codes are shown in Figure 5.

P	r	d_{free}	$N_{d_{free}}$	$I_{d_{free}}$	Generator Sequences (Octal)
1	2/3	3	2	4	[6,2,6 ; 2,4,4]
2	2/3	4	23/2	66/2	[6,4,6 ; 2,6,0] [6,4,0 ; 2,6,6]

Table 4: The $(3,2,1)$ time invariant MFD code and the $(3,2,1,2)$ periodic time varying convolutional code found by Palazzo.

By randomly searching for period $P = 4$ time varying convolutional codes, Lee[3] found a code with memory 4 and rate 1/2 which achieves Heller's upper bound. It is the first convolutional code with memory 4 that achieves the Heller bound. This code together with the $(2,1,4)$ time invariant MFD code are listed in Table 5. The simulation results for these codes are shown in Figure 6.

P	r	d_{free}	$N_{d_{free}}$	$I_{d_{free}}$	Generator Sequences (Octal)
1	1/2	7	2	4	[23,35]
2	1/2	8	49/4	178/4	[37,20] [37,35] [33,25][33,25]

Table 5: The $(2,1,4)$ time invariant MFD code and the $(2,1,4,4)$ periodic time varying convolutional code found by Lee.

Studying the performance of the codes found by Palazzo, it can be seen from Figure 4 and Figure 5 that the periodic time varying codes show a small performance improvement over the time invariant MFD codes. In Figure 4 all the periodic time varying codes are better than the time invariant MFD code in terms of P_b and all these codes have smaller $N_{d_{free}}$ and $I_{d_{free}}$ than that of the time invariant

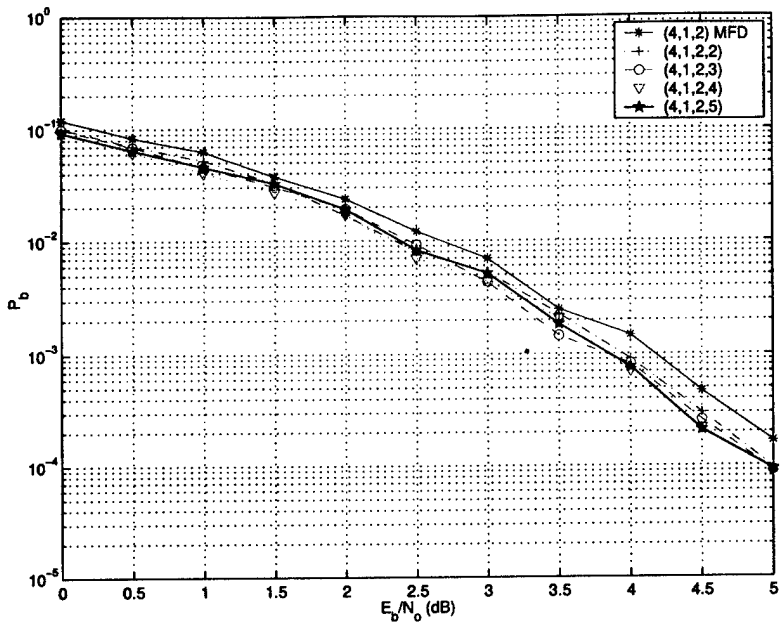

Figure 4: Simulation results of the time invariant MFD code and the $(4, 1, 2, P)$ codes found by Palazzo.

MFD code. In Figure 5, the $(3, 2, 1, 2)$ periodic time varying code with $d_{free} = 4$ has better P_b for almost all SNR's than the $(3, 2, 1)$ time invariant MFD code with $d_{free} = 3$.

However, from Figure 3, the $(2, 1, 4, P)$ codes found by Mooser do not have better performance than the $(2, 1, 4)$ time invariant MFD code even though the $(2, 1, 4, P)$ codes are better in terms of $N_{d_{free}}$ and $I_{d_{free}}$. Also, from Figure 6, it can be seen the the $(2, 1, 4, 4)$ code with $d_{free} = 8$ has almost the same performance as the $(2, 1, 4)$ time invariant MFD code with d_{free}. Hence, periodic time varying codes with larger d_{free} than the comparable time invariant codes, or with the same d_{free} but smaller $N_{d_{free}}$ and $I_{d_{free}}$ are not guaranteed for better performance at low and moderate SNR's than the comparable time invariant codes. In fact, the distance spectrum is the main factor in determining the error probability when Viterbi decoding is used for a convolutional code.

The code search in this thesis has two goals. First, to try to demonstrate the argument of Johannesson [4] that the Heller bound can be applied to time varying codes but the Griesmer bound can not. From Figure 7, it can be seen that there is a gap between the Heller bound and the Griesmer bound for memories 5, 12, and so on. For example, the Heller bound for rate 1/2 memory 5 time varying codes is 9 while the Griesmer bound is 8. A search for $(2, 1, 5, P)$ periodic time varying codes with $d_{free} = 9$ is performed in this thesis. Memory 5 is the smallest for rate 1/2 codes where the Griesmer bound does not agree with the Heller bound. The second goal of the search is to find new codes of rate 1/2 and memory greater than 5 with larger free distance than the comparable time invariant MFD codes. The first interesting class of codes is the $(2, 1, 7, P)$ codes with $d_{free} = 11$.

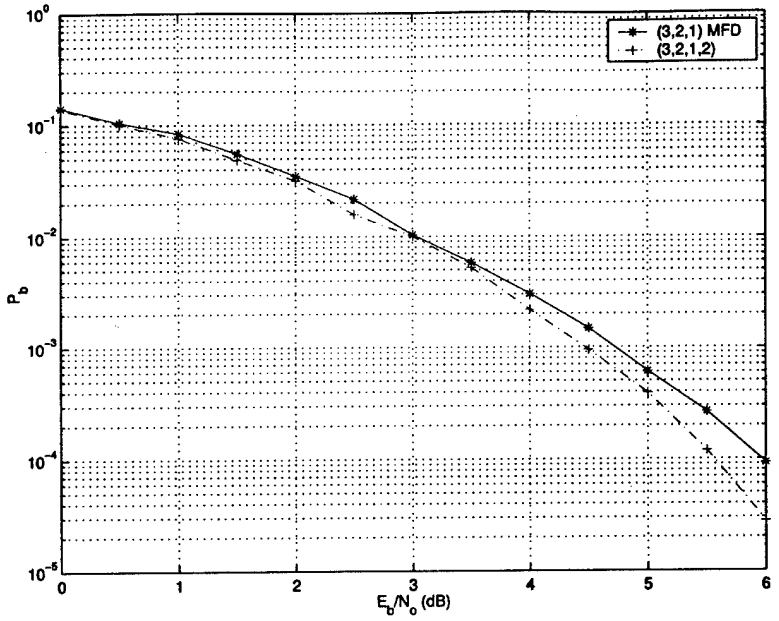

Figure 5: Simulation results of the time invariant MFD code and the $(3, 2, 1, 2)$ codes found by Palazzo.

3.3 Search Technique and Result

So far there has been little success in finding good convolutional codes in terms of large free distance by algebraic methods. Most codes are found by computer search [5, 6, 7, 8, 9]. A computer aided code search based on the FAST algorithm [10] for computing the distance spectrum of time invariant convolutional codes is used in this thesis. With a few modifications, FAST can be used to find the free distance and the distance spectrum of a periodic time varying code.

3.3.1 The FAST Algorithm for Time Invariant Convolutional Codes

The FAST algorithm for finding the distance spectrum was developed by Cedervall and Johannesson [10, 4]. It utilizes the distance profile to dramatically reduce the search to relatively small number of codewords. Recall that the distance profile is the first $m+1$ orders of the column distance. This algorithm can easily be modified to find the free distance of a time invariant code and to determine the number of nonzero information bits and path length of a codeword.

Let $n_{d_{free}+i}$ denote the number of paths of weight $d_{free}+i$ which depart from the all zero path at the root node in the code trellis and do not reach the zero state until their termini. That is, $n_{d_{free}+i}$ is the number of codewords with weight $d_{free}+i$. $n_{d_{free}+i}$ is called the $(i+1)$th *spectral component.* The sequence

$$n_{d_{free}+i}, \qquad i = 0,\ 1,\ 2,\ \ldots,$$

is called the *distance spectrum* of the code.

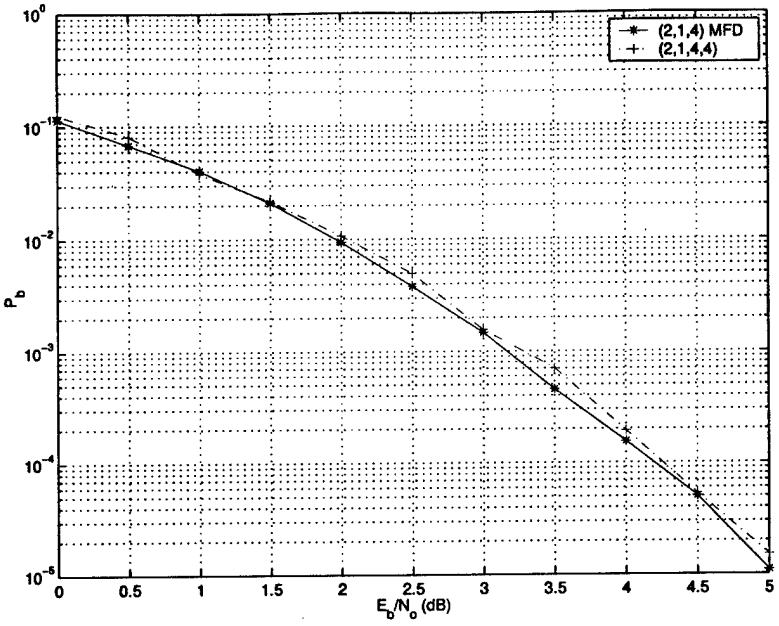

Figure 6: Simulation results of best time invariant codes and (2,1,4,4) codes found by Lee.

To compute the distance spectrum for a time invariant convolutional code encoded by a feedforward encoder, the FAST algorithm exploits the linearity of the code and counts the number of weight d codewords with $\mathbf{u}_0 \neq \mathbf{0}$ diverging from the zero state and remerging for the first time at the zero state. As before, $\mathbf{u}_t$ and $\mathbf{v}_t$ denote the input and output at time instant t, respectively. This algorithm performs a sequential search in the code trellis to find a sequence of weight d and, hence, it is essentially a progressive trellis search. For simplicity, the discussion is limited to the case of rate 1/2. The extension to rate k/n is straightforward.

For an arbitrary node at depth t in the code trellis, let $\mathbf{W}_t$, where $\mathbf{W}_t > d$, be the accumulated total weight of a certain path produced by $t-1$ inputs. A subtrellis is defined as the remaining forward trellis connecting to this node. For each subtrellis stemming from this node with weight d, when it terminates at the zero state it has to *spend* exactly weight $(d - \mathbf{W}_t)$. Hence, each node is labeled with the current *state* of the encoder and the *remaining weight*, that is, $\mathbf{W} = d - \mathbf{W}_t$. When the node is at the zero state and its weight is $\mathbf{W} = 0$, a path with weight d is found.

Let $\mathbf{S}_t = (\mathbf{u}_{t-1}\ \mathbf{u}_{t-2}\ \cdots\ \mathbf{u}_{t-m})$ denote the *state* of the encoder and let $\mathbf{u}_t = 0$ for $t < 0$. This definition is true only for feedforward encoders, since at time t the content of the ith stage of a shift register is just the input at time $t-i$. For feedback encoders[1], the state depends not only on the current state, but also on the combination of the previous inputs and the output of the encoder. Therefore, the state of a feedback encoder is not $(\mathbf{u}_{t-1}\ \mathbf{u}_{t-2}\ \cdots\ \mathbf{u}_{t-m})$. There are two successor states from each state, namely, $\mathbf{S}^0_{t+1} = (0,\ \mathbf{u}_{t-1}\ \cdots\ \mathbf{u}_{t-m-1})$ and $\mathbf{S}^1_{t+1} = (1,\ \mathbf{u}_{t-1}\ \cdots\ \mathbf{u}_{t-m-1})$, corresponding to $\mathbf{u}_t$ equal to zero and one, respectively. To simplify the notation, t will be ignored in the sequel. Let w^0 and w^1 denote the branch weights stemming from a node. Using these branch weights together with the current node weight $\mathbf{W}$, the two successor node weights can be expressed

[1]encoders with feedback connections

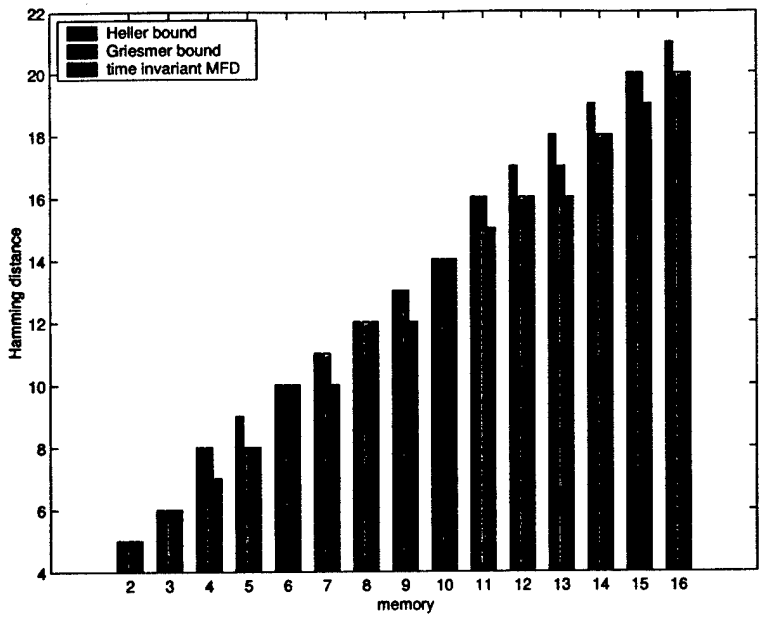

Figure 7: Heller bound, Griesmer bound and d_{free} of the time invariant MFD codes.

as

$$\mathbf{W}^0 = \mathbf{W} - w^0 \quad \text{and} \quad \mathbf{W}^1 = \mathbf{W} - w^1.$$

This is illustrated in Figure 8.

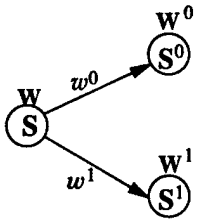

Figure 8: Successor nodes at time t.

When searching for a path in the code trellis with a given weight, a subtrellis will be explored if and only if either of the new node weights, $\mathbf{W}^0$ or $\mathbf{W}^1$, is nonnegative and if the state of the new node, $\mathbf{S}^0$ or $\mathbf{S}^1$, differs from the zero state. A negative node weight means that paths stemming from the current node have weight smaller than d and therefore need not be extended. When the state of the new node is zero, it means the path has reached its terminus and further extension is not necessary. Without loss of generality, priority is given to the zero branch whenever a selection has to be made between two new possible nodes. With this background, a straightforward algorithm for determining the number of paths of a given weight d can be formulated as follows.

Let the root node be defined as the node with state $\mathbf{S} = (0\ 0\ \ldots\ 0)$ and weight $\mathbf{W} = d$. Corresponding to the input zero and one, the two successor nodes are the node with state $\mathbf{S}^0 = (0\ 0\ \ldots\ 0)$ and weight $\mathbf{W}^0 = d$, and the node with $\mathbf{S}^1 = (1\ 0\ \ldots\ 0)$ and weight $\mathbf{W}^1 = d - d_{c,0}$, where $d_{c,0}$ is the 0th order column distance. For rate 1/2 codes, there is $w^1 = d_{c,0}$. Selecting the first node results

in the all zero path search or the replica of the search when the second node is selected. Therefore, the search starts at the second node with state $\mathbf{S}^1 = (1\ 0\ \ldots\ 0)$ and weight $\mathbf{W} = d - d_{c,0}$ and moves forward in the code trellis. If the state of current node is $\mathbf{S} = (0\ 0\ \ldots\ 1)$ and the new node corresponding to zero input has weight $\mathbf{W}^0 = 0$ and state $\mathbf{S}^0 = (0\ 0\ \ldots\ 0)$, then the path counter n_d is increased by 1. n_d is used to keep track of the number of paths with weight d. If the new node weight is negative or if the new node is in its zero state, then the search moves backward. Thus, all the previous information symbols have to be stored so that the algorithm can move back until a new "one" branch with a nonnegative node weight is found. The search then moves forward again. A *stop condition* appears when the search reaches the root node.

This basic algorithm is very time consuming. Through an example, the efficiency of the FAST algorithm will be explained. A $(2, 1, 3)$ time invariant code is used for this example. The generator

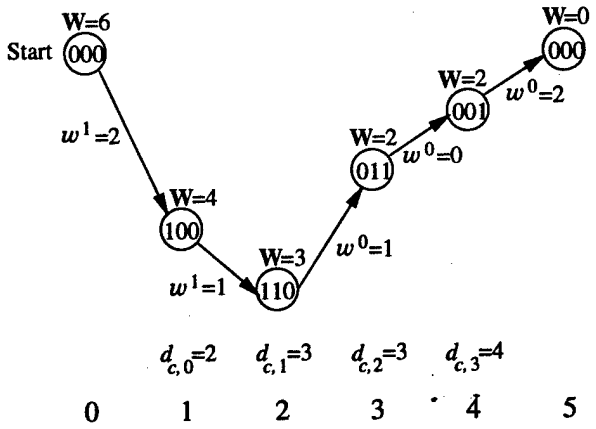

Figure 9: An example of a weight d_{free} path

sequences of this code are $[\mathbf{g}^{(1)}, \mathbf{g}^{(2)}] = [74, 54]$. It has a distance profile of $\mathbf{d} = [2,\ 3,\ 3,\ 4]$. In Figure 9, the part of its trellis which contains the weight $d_{free} = 6$ path is shown. This path corresponds to the information sequence $\mathbf{u}_{[0,\,5)} = (1\ 1\ 0\ 0\ 0)$ and to the encoded sequence $\mathbf{v}_{[0,\,5)} = (11\ 01\ 01\ 00\ 11)$. Since the column distance is the *minimum* of the Hamming weights of the paths with $\mathbf{u}_0 = 1$, the distance profile can be used as a *lower* bound on the *decrease* of the node weight along the path. In general, the distance profile gives the minimum accumulated path weights for each of the first $m + 1$ time units. Since the node weight is computed by the total weight minus the accumulated path weight and the accumulated path weight is lower bounded by the distance profile, the decrease of the node weight, which is the total weight minus node weight, is lower bounded by the distance profile.

For example, at time unit 1, from Figure 9, the node weight is $\mathbf{W} = 4$ due to the branch weight $w^1 = 2$. The difference between the total weight $d = 6$ and the node weight 4 is 2. Hence, the node weight decreases by 2. At time unit 2, the node weight goes down to $\mathbf{W} = 3$ because of the branch weight $w^1 = 1$. Therefore, the node weight decrease by 3. The difference between the node weight and the total weight for time units 3 and 4 are 4 and 4, respectively. The decrease of the node weights for the four units is $[2,\ 3,\ 4,\ 4]$. $\mathbf{d}$ is a tight lower bound for time units 1, 2 and 4.

Before further discussion of the algorithm, let us first look at how to move backward in the trellis. For the two encoders shown in Figure 10, they have the same connection except the input of encoder

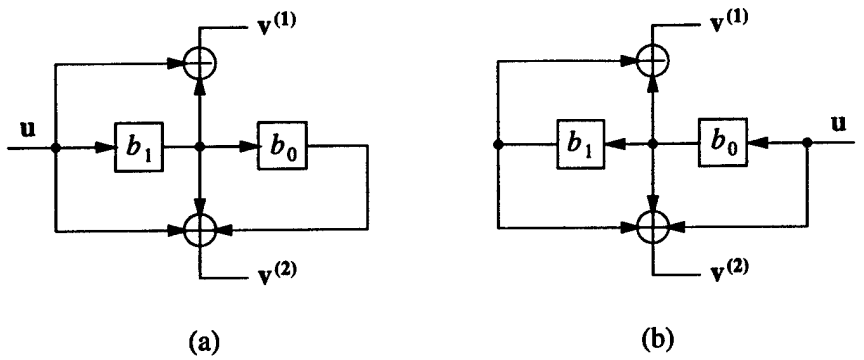

Figure 10: Encoders of a (2,1,2) time invariant code with left side input and right side input.

(a) enters from left side while the input of encoder (b) enters from right side. The trellises of these two encoders are shown in Figure 11. Comparing these two trellis, it is easy to see that the two encoders share the same trellis structure with the same output branch label. The only difference is that one moves from left to right (forward) and the other one moves from right to left (backward). Therefore, in order to travel in the opposite direction in a trellis, the input should go into the encoder from the other side. For example, let the $\mathbf{u}_{[0,6)} = (1\ 0\ 1\ 0\ 0)$. The output of encoder (a) is $\mathbf{v}_{[0,6)} = (11\ 11\ 10\ 11\ 01)$ and the output of encoder (b) is $\mathbf{v}_{[0,6)} = (01\ 11\ 10\ 11\ 11)$. Note the order of theses two outputs are reversed due to the opposite travel direction. These results can be obtained by following the two red paths in Figure 11.

If the search traverses this path in the opposite direction, it will get the same total weight but different node weights. The node weight can be explained as the weight that needs to be spent on the remainder of this path (backward). It is the decrease of node weight when moving forward. Thus, the node weights are lower bounded by the distance profile when moving backward in the trellis. In Figure 12, the distance profile is used as a lower bound on the node weight along the path. Notice that if a node has weight less than this bound, then every path leading backward to the zero state will give a negative node weight at the root node. For example, if the node weight in state (001) is less than $d_{c,3} = d_{c,m} = 4$, this node will not be extended when traversing the trellis backward. This is because, from the state (001) in that path, at least weight 4 needs to be spent for the next 4 steps to reach the all zero state. More generally, the weight of a backward path stemming from a node in state $\mathbf{S} \neq (00\ldots0)$, starting with a one branch and eventually leading to the root node (zero state), is lower bounded by $d_{c,m}$.

The use of the distance profile as a lower bound on the node weights works for every path from the end to the root node. Moving backward from state $\mathbf{S}$, two possible states $\mathbf{S}^{-0}$ and $\mathbf{S}^{-1}$ will be reached, where The minimum weight of the backward paths stemming from the states $\mathbf{S}^{-0}$ and

$$\begin{aligned}
\mathbf{S} &= (?\ldots??1\ \underbrace{0\ldots00}_{l-1 \text{ zeros}}) \\
\mathbf{S}^{-0} &= (?\ldots?1\underbrace{00\ldots00}_{l \text{ zeros}}) \\
\mathbf{S}^{-1} &= (?\ldots?1\underbrace{00\ldots01}_{l-1 \text{ zeros}})
\end{aligned}$$

$\mathbf{S}^{-1}$ are lower bounded by $d_{c,m-l-1}$ and $d_{c,m-1}$, respectively. For the backward paths stemming

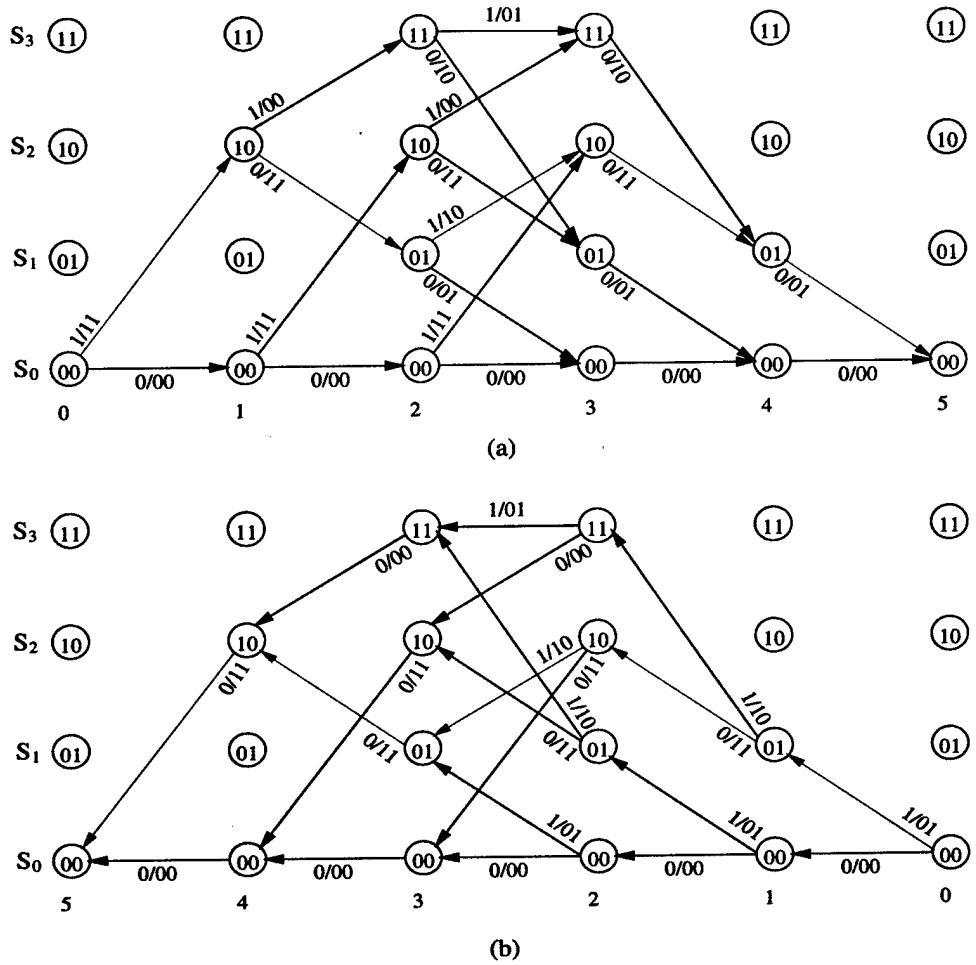

Figure 11: Trellis of the two encoders in Figure 10.

from the state $\mathbf{S}^{-0}$, at least $m - l$ more steps are needed for the paths to reach the zero state, which means at least weight $d_{c,m-l-1}$ needs to be spent. For the backward paths stemming from the state $\mathbf{S}^{-1}$, at least m more steps are needed for the paths to get to the zero state and at least weight $d_{c,m-1}$ needs to be spent. Therefore, $d_{c,m-l-1}$ and $d_{c,m-1}$ become lower bounds for the node weights at states $\mathbf{S}^{-0}$ and $\mathbf{S}^{-1}$.

Moving backward in the trellis is not convenient because it requires input from the other side of the encoder. In fact, moving forward in the trellis generated by the *reversed* generator sequences of the original generator sequences is equivalent to moving backward in the trellis of the original code. Figure 13 shows the encoder with a reversed generator sequences of the encoder (a) in Figure 10. Its trellis is shown in Figure 14. Let $\mathbf{u}_{[0,6)} = (1\ 0\ 1\ 0\ 0)$. The output of this "reversed" encoder is $\mathbf{v}_{[0,6)} = (01\ 11\ 10\ 11\ 11)$, which is the same as the output of encoder (b) in Figure 10 whose trellis moves backward. Therefore, instead of moving backward in the trellis, the generator sequences can be *reversed* and the search can move forward in the corresponding trellis and the distance profile (of the original generator sequences) can be used to effectively limit the part of the trellis that must be explored.

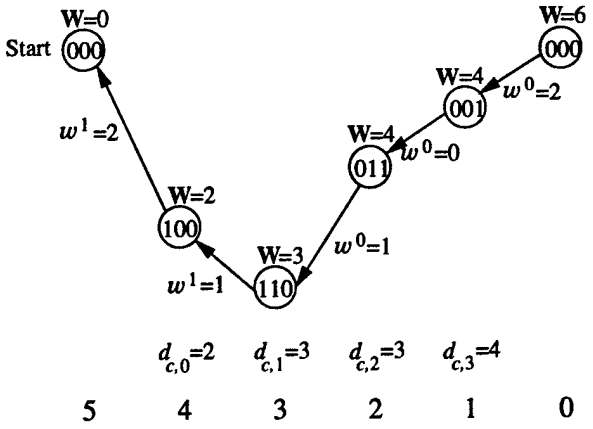

Figure 12: The weight d_{free} path traversed backward

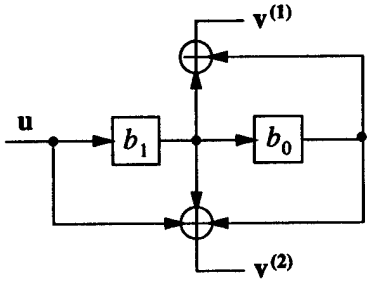

Figure 13: Encoder of a (2,1,2) time invariant code with a reversed generator sequences of encoder (a) shown in Figure 10.

The FAST algorithm can now be described as follows. Suppose the distance spectrum of code with generator sequences $[\mathbf{g}^{(1)}, \mathbf{g}^{(2)}]$ and distance profile $\mathbf{d}$ is going to be determined. Let $[\tilde{\mathbf{g}}^{(1)}, \tilde{\mathbf{g}}^{(2)}]$ denote the generator sequence for the *reversed* time invariant convolutional code. To calculate the ith spectral component, start at state $\mathbf{S} = (10\ldots0)$ with weight $\mathbf{W} = d_{free} + i - d_{c,0}$ in the code trellis generated by $[\tilde{\mathbf{g}}^{(1)}, \tilde{\mathbf{g}}^{(2)}]$. This weight will be *reduced* by the weight of the branches that the search traverses when the code trellis is searched for nodes with both node weight and state equal to zero. For each explored node, the column distance $d_{c,m-l-1}$ or $d_{c,m-1}$ will be used to lower bound the weight of any path leading to a zero state. If the current weight is less than this bound, a nonzero state with zero or negative weight will always by reached. Hence, it is only necessary to extend a node if the node weight is larger than or equal to this bound.

If both successor nodes are achievable, then the "zero" branch is selected and the "one" branch node (state S^1 and weight W^1) is pushed on a stack. Thus, the weight of this node will not be calculated twice while moving back. The complete FAST algorithm is described as follows and the flow chart is shown in Figure 15. Notice that w^i is calculated using the reversed generator sequences.

F1 (Initialize) Set $l \leftarrow 1$, $n_d \leftarrow 0$, $\mathbf{W} \leftarrow d - d_{c,0}$, and $\mathbf{S} = [1,\ 0, \cdots,\ 0]$.

F2 (Next nodes) Calculate $\mathbf{S}^0$, $\mathbf{S}^1$, $\mathbf{W}^0$ and $\mathbf{W}^1$. If $l < m$, go to **F6**.

F3 (Return to zero) If $\mathbf{W}^0 = 0$, set $n_{d-\mathbf{W}^0} \leftarrow n_{d-\mathbf{W}^0} + 1$.

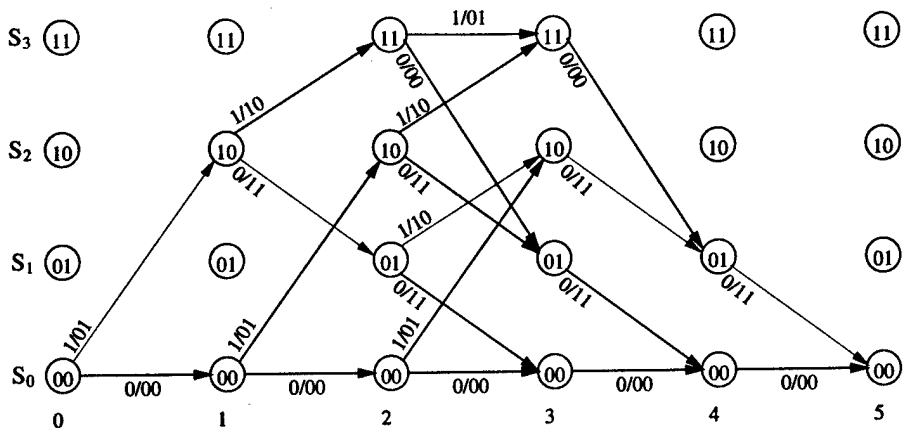

Figure 14: Trellis of the encoders in Figure 13.

F4 (Forward on "one" branch?) If $\mathbf{W}^1 < d_{c,m-1}$ or $\mathbf{W} < d_{c,m}$, go to F5. Otherwise, select "one" branch node and set $l \leftarrow 1$. Go to **F2**.

F5 (From stack?) If stack is empty, the algorithm terminates with the result in n_d. Otherwise, select the node from the stack and set $l \leftarrow 1$. Go to **F2**.

F6 (Forward on "zero" branch?) If $\mathbf{W}^0 < d_{c,m-l-1}$, go to **F4**.

F7 (Save "one" branch node?) If $\mathbf{W}^1 \geq d_{c,m-1}$ and $\mathbf{W} > d_{c,m}$, save the "one" branch node. In any case, select ' 'zero" branch node and set $l \leftarrow l + 1$. Go to **F2**.

The algorithm described above assumes the free distance is known. This algorithm can be modified to find the free distance of a code when it is unknown as follows.

Initialize d to be sufficiently large, e.s., 1000. If $d > d_{free}$, the search will reach a zero state with a positive node weight $\mathbf{W}^0$, i.e., d_{free} is at most $d - \mathbf{W}^0$. Hence all node weights in the stack can be reduced by $\mathbf{W}^0$ (adjust stack) and the path counter reset to 1. When the search stops, $d = d_{free}$. The modifications of **F3** in the algorithm are described below and the modified flow chart is shown in Figure 16.

F3 (Return to zero) If $\mathbf{W}^0 = 0$, set $n_d \leftarrow n_d + 1$; else if $\mathbf{W} > 0$, set $d \leftarrow d - \mathbf{W}^0$, $\mathbf{W}^1 \leftarrow \mathbf{W}^1 - \mathbf{W}^0$, $\mathbf{W} \leftarrow \mathbf{W} - \mathbf{W}^0$, $n_d \leftarrow 1$ and adjust stack.

3.3.2 Modification to FAST for Periodic Time Varying Codes

In order to make FAST work for periodic time varying codes, there are three necessary modifications. First, since one periodic time varying encoder consists of P time invariant encoders (as shown in Figure 1), all the generator sequences of the P time invariant encoders have to be reversed instead of just one. Second, the distance profile used must be modified to be

$$\mathbf{d}^p = [d^p_{c,0}, d^p_{c,1}, \ldots, d^p_{c,m}]$$

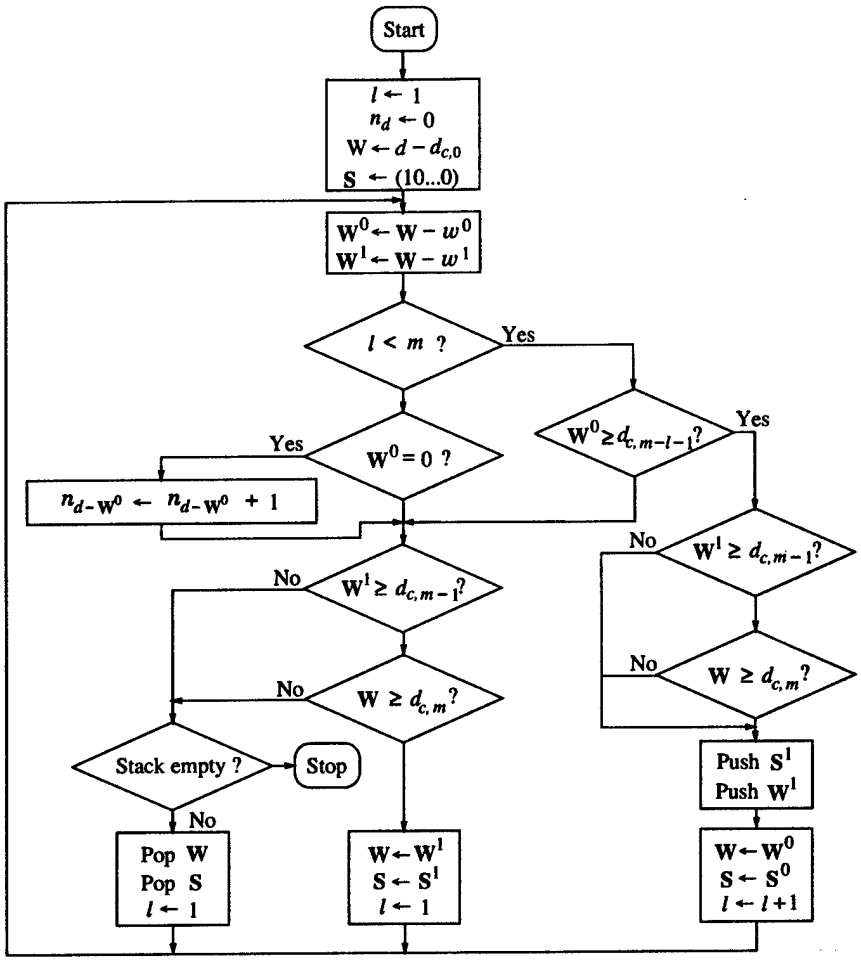

Figure 15: Flow chart of the FAST algorithm.

$$= \quad [\min_j d^p_{c,0}(j),\ \min_j d^p_{c,1}(j),\ \ldots, \min_j d^p_{c,m}(j)] \qquad j = 1,\ 2,\ \ldots,\ P$$

Third, the time invariant code C_j used at each time instant must be recorded.

Recall that for time invariant codes, moving forward in the trellis generated by the *reversed* code is equivalent to moving backward in the trellis of the original code. This is also true for periodic time varying codes. However, moving backward in the trellis of a periodic time varying code $C_1C_2\cdots C_{P-1}C_P$ is equivalent to moving forward in the trellis of $C_PC_{P-1}\cdots C_2C_1$. A counter j is introduced in the algorithm to keep track of the current time invariant code.

Let $w^0(j)$ and $w^1(j)$ denote the branch weight generated by code C_j with input "0" or "1", respectively. Note that in order to find all possible paths of weight d for periodic $C_1C_2\cdots C_{P-1}C_P$ code, FAST has to be run P times to check the P circular shifts, e.s., $C_2C_3\cdots C_PC_1$. Steps **F1** and **F2** are modified as:

F1 (Initialize) Set $i \leftarrow 1$, $l \leftarrow 1$, $n_d \leftarrow 0$, $\mathbf{W} \leftarrow d - d^p_{c,0}$, and $\mathbf{S} = [1,\ 0, \cdots,\ 0]$.

F2 (Next nodes) Calculate $j = (j+P-1) \bmod P$, $\mathbf{S}^0$, $\mathbf{S}^1$, $\mathbf{W}^0 = \mathbf{W} - w^0(j)$ and $\mathbf{W}^1 = \mathbf{W} - w^1(j)$.

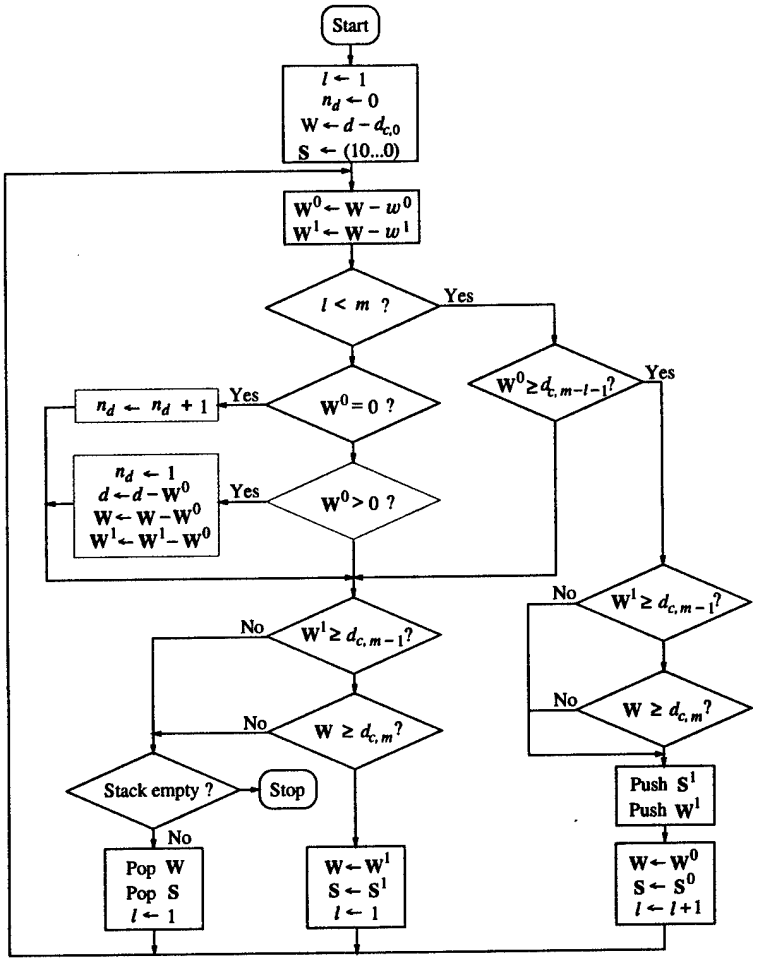

Figure 16: Flow chart of the FAST algorithm when d_{free} is unknown.

If $l < m$, go to **F6**.

Figure 17 shows the flow chart of the modified FAST algorithm for periodic time varying codes. Figure 18 shows the flow chart of FAST for periodic time varying codes when d_{free} is unknown.

3.3.3 Search Technique

Before running the code search program, the ensemble of periodic time varying codes needs to be generated. First, the ensemble of $(2, 1, m)$ time invariant codes is generated. For $(2, 1, m)$ codes, there are two generator sequences, $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$. Each generator has $m + 1$ bits that can be "0" or "1". There are 2^{m+1} possible values for each generator sequence. Therefore, there is a total $2 \times 2^{m+1}$ possible codes. However, time invariant codes generated in this way may not have memory m. For example, to generate a $(2, 1, 2)$ code, the $\mathbf{g} = [4, 2] = [110, 010]$ is a valid combination, but no longer a valid $(2, 1, 2)$ code. Because the last position of $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$ are all "0". To solve this problem, the following restriction is imposed on $\mathbf{g}^{(1)}$. The mod 2 adder has connections to the present input

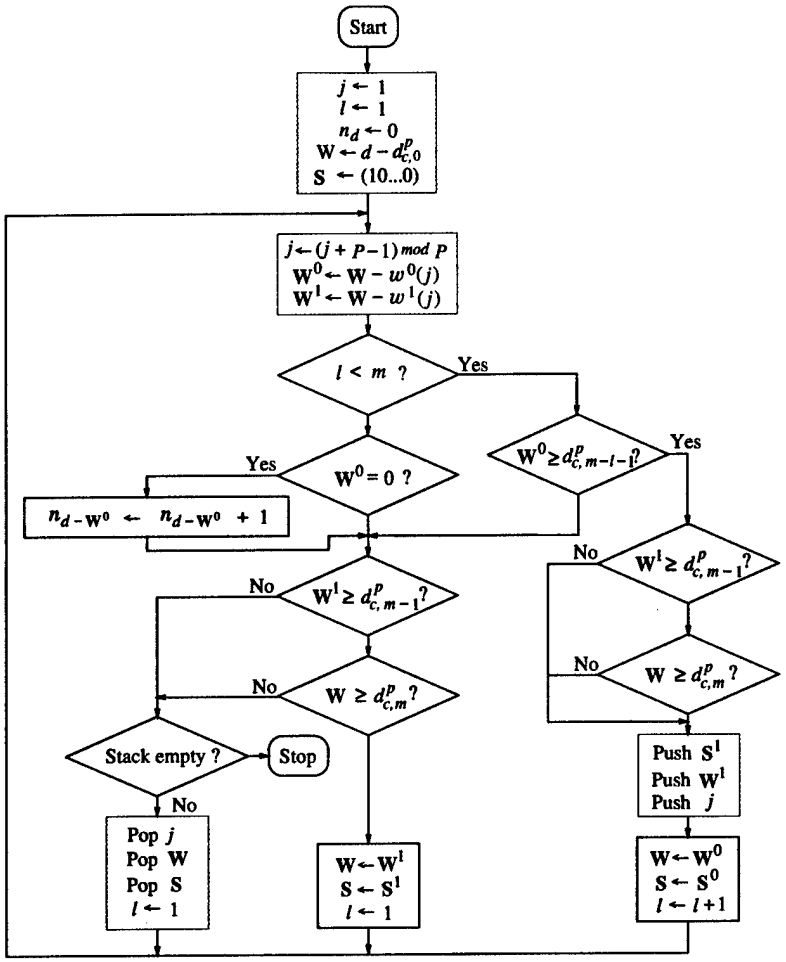

Figure 17: Flow chart of the modified FAST algorithm.

and to the oldest input, that is, $\mathbf{g}^{(1)}$ has "1's" in the first and last positions. Therefore, there are 2^{m-1} possible value for $\mathbf{g}^{(1)}$. $\mathbf{g}^{(2)}$ is constructed without the above restriction and still has 2^{m+1} values. Next, periodic time varying codes are generated based on the time invariant code ensemble. If the period is 2, then two time invariant code ensembles are chosen. The period 2 time varying codes are generated by make all possible combinations of these two ensembles. To generate period 3 time varying codes, pick 3 time invariant code ensembles and do the combinations. For period P codes, P time invariant code ensembles are used to make the combinations.

As can be seen from the above discussion, the number of possible codes increase exponentially with the period. Eliminating the equivalent codes from the code ensemble becomes very important in order to make the code search feasible. By studying periodic time varying codes, a criterion is found to get rid of a large amount of codes.

First, consider a period 2 code in the order of C_1C_2, then the code in the order C_2C_1 will be the same code as C_1C_2. Next, for a period 3 code in the order $C_1C_2C_3$, the equivalent codes are $C_2C_3C_1$ and $C_3C_1C_2$. For the period 3 code, the last two codes are just circular shift of the first one. From the discussion of the FAST algorithm, the algorithm itself checks all the circular shifts

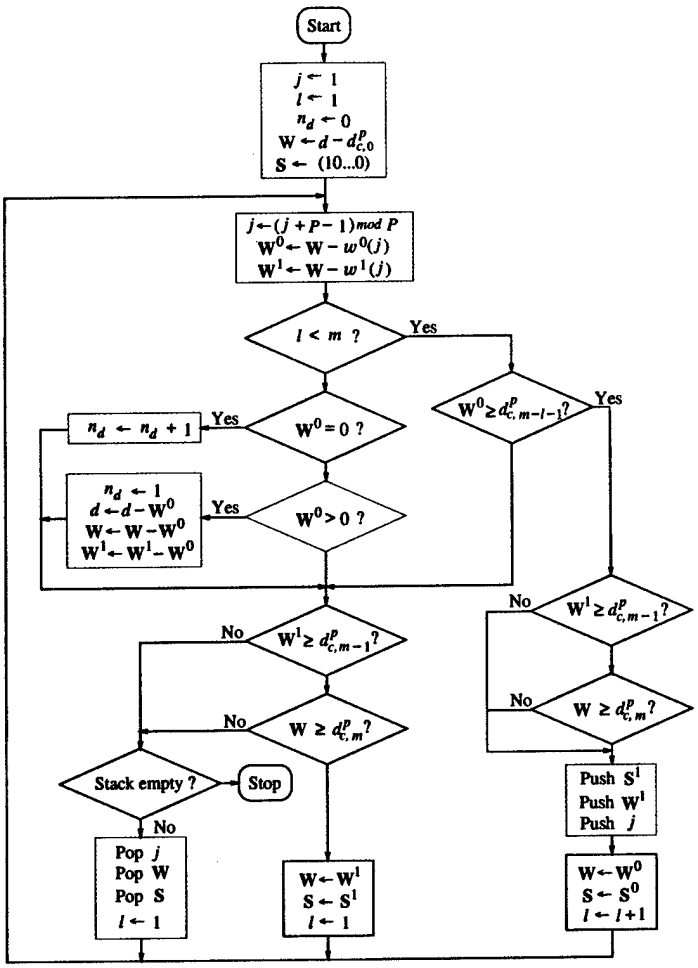

Figure 18: Flow chart of the modified FAST algorithm when d_{free} is unknown.

and these three codes have the same set of circular shifts. Hence, the last two are equivalent to the first one. The same principle can be use to find equivalent codes for any period. That is, codes that are generated by a circular shift right or left j times, $j = 0, \ldots, P$, are equivalent codes. Using this method, about 25% codes can be eliminated.

As mentioned before, the FAST algorithm performs a progressive trellis search. If the search goes into a path that can not gain any weight, it will keep going and never stop. This happens when a catastrophic encoder is being searched. Since the generated code ensemble has a lot catastrophic codes, these catastrophic codes should be eliminated before the search starts or the FAST algorithm needs to be modified to detect the catastrophic code. A discussion on checking the catastrophic condition will be presented in Chapter 4 based on a transition matrix analysis. This can be used to eliminate the catastrophic encoders before the search starts. A trivial modification to FAST to identify a catastrophic encoder is to add a counter to keep track of the path weight. If the weight does not increase for a long time, then the search should stop.

3.3.4 Search Results

The search for $(2,1,5,P)$ codes with $P = 1$, 2, 3, and 4 are exhaustive. The search for $(2,1,5,5)$ and $(2,1,7,2)$ codes are not exhaustive because of the huge code ensemble. For these two codes, codes are picked randomly from the overall ensemble of each code.

Code	C_1	C_2	d_{free}	$N_{d_{free}}$	$I_{d_{free}}$
1	(367 255)	(323 247)	11	9	19
2	(335 257)	(323 247)	11	13/2	47/2
3	(375 267)	(323 247)	11	17/2	69/2

Table 6: $(2,1,7,2)$ Codes With Free Distance 11.

In Table 6, all of the $(2,1,7,2)$ codes with free distance 11 are listed. The C_1 and C_2 column headings represent the two time invariant codes used in the period 2 time varying code. The generators are written in an octal form. For each code, $N_{d_{free}}$ and $I_{d_{free}}$ are also presented.

Code		$d = 10$	11	12	13	14	15	16	17
1	N_d	0	9	33/2	14	45	160	363	1517/2
	I_d	0	38	81	91	338	1352	3312	15001/2
2	N_d	0	13/2	16	26	54	255/2	639/2	791
	I_d	0	47/2	83	165	375	2073/2	2857	7707
3	N_d	0	17/2	16	20	56	136	655/2	794
	I_d	0	69/2	84	138	839/2	1145	3002	7843
MFD	N_d	1	6	12	26	52	132	317	730
	I_d	2	22	60	148	340	1008	2642	6748

Table 7: $(2,1,7,2)$ Distance Spectrum. Notice that the first distance with nonzero spectrum is d_{free}.

Table 7 lists the distance spectrum of all $(2,1,7,2)$ codes with free distance 11, from $d = 10$ to $d = 17$. N_d is the total number of codewords of weight d divided by period P and I_d is the total information weight of all codewords of weight d divided by period P. The distance spectrum of the $(2,1,7)$ time invariant MFD code with $\mathbf{g} = [712, 476]$ and free distance 10 are also listed in Table 7. Compared with this time invariant MFD code, the three $(2,1,7,2)$ codes are good in terms of free distance. However, except for distance 10, the spectrum of the time invariant MFD code is better (smaller) than all three periodic time varying codes.

Unfortunately, after an extensive search of $(2,1,5,P)$ codes with $P = 1,2,3,4$, and 5, a code with $d_{free} = 9$ was not found. Since the ensemble of codes with period greater than 5 becomes too large, these codes were not searched in this thesis. It is still possible that a $(2,1,5,P)$ periodic time varying codes with $d_{free} = 9$ exists.

3.4 Simulation Results

In this section, simulation results for the three periodic time varying codes with $d_{free} = 11$ are presented. These results are compared with the simulation results for the $(2, 1, 7)$ time invariant MFD code with $\mathbf{g} = [712, 476]$ [11].

In these simulations, more than 100 bit errors are counted for each simulation run in order to reduce the variance of the estimate. The bit error rate is plotted versus signal to noise ratio (E_b/N_o) in all the figures. Here, E_b represents the energy per information bit and N_o is the single sided power spectral density of the white noise. Binary Phase Shift Keying (BPSK) signaling is used over an additive white Gaussian noise(AWGN) channel and soft decisions are used in the decoder. Figures 19, 20 and 21 show the simulation results of the three $(2, 1, 7, 2)$ periodic time varying codes, respectively.

An analytical BER given by

$$P_b(E) \leq \frac{1}{k} \sum_{d=d_{free}}^{\infty} I_d P_d, \tag{1}$$

where I_d is the total number of nonzero information bits on all weight d paths and P_d is the probability that decoder selects an incorrect path with Hamming distance d from the correct path is also shown in the figures. For the AWGN channel with BPSK modulation and a soft decision decoder

$$P_d = Q\left(\sqrt{2dRE_b/N_o}\right) \tag{2}$$

where $Q(\cdot)$ is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\beta^2/2} d\beta.$$

In practice, the summation in (1) is truncated to a finite number of terms to obtain an approximation on P_b. In Figures 19, 20 and 21, 16 terms are used. The 16 I_d terms used in Figure 19 are $\frac{1}{2}\{$ 76, 162, 182, 676, 2704, 6624, 15001, 41446, 112993, 293480, 744716, 1904866, 4920721, 12584340, 31989423, 81053650 $\}$. The 16 I_d terms used in Figure 20 are $\frac{1}{2}\{$ 47, 166, 330, 750, 2073, 5714, 15414, 39728, 104476, 271160, 688352, 1759408, 4491857, 11429308, 28964105, 73067376 $\}$. The I_d used in Figure 21 are $\frac{1}{2}\{$ 69, 168, 276, 839, 2290, 6004, 15686, 41239, 109246, 281140, 719852, 1843439, 4688815, 11885233, 30098536, 76072907 $\}$.

The simulation results show that even though the periodic time varying convolutional codes have larger free distance than the time invariant codes, they do not outperform the time invariant codes for low and moderate E_b/N_o's. From Figure 19, it can be seen that the two performance curves are interweaved with each other. Hence, it is hard to say that the time invariant code is better than the periodic time varying code or the time varying code is better than the time invariant code. This situation is also true in Figures 20 and 21.

This can be explained by using the distance spectrum together with the union bound[12]. That is, the performance is determined by the overall distance spectrum, not only by the free distance term. The $(2, 1, 7)$ time invariant code has $d_{free} = 10$ which is smaller than the periodic time varying codes, but the $N_{d_{free}}$ and the $I_{d_{free}}$ are fairly small, only 1 and 2. Comparing the other distance spectrum terms with those of the periodic time varying codes, this time invariant code is

better for almost all distances. Therefore, in terms of distance spectrum, the three $(2, 1, 7, 2)$ codes are denser than the time invariant code. The advantage the three $(2, 1, 7, 2)$ periodic time varying convolutional codes gain from their larger free distance is offset by their relatively dense distance spectrum.

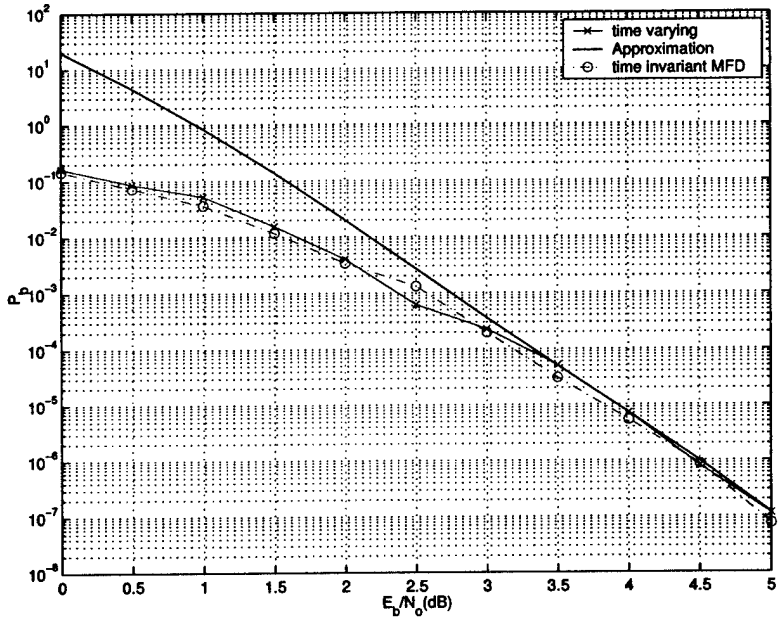

Figure 19: Simulation of (2,1,7,2) No.1.

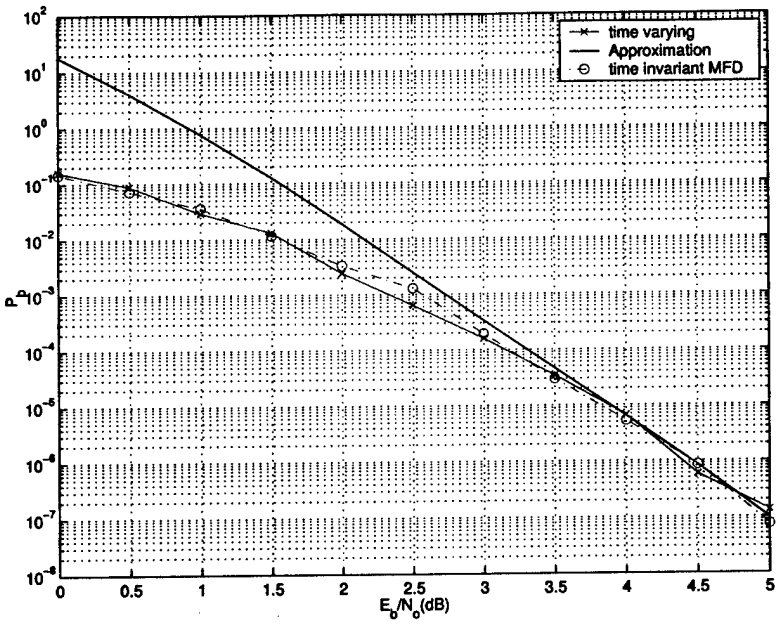

Figure 20: Simulation of (2,1,7,2) No.2.

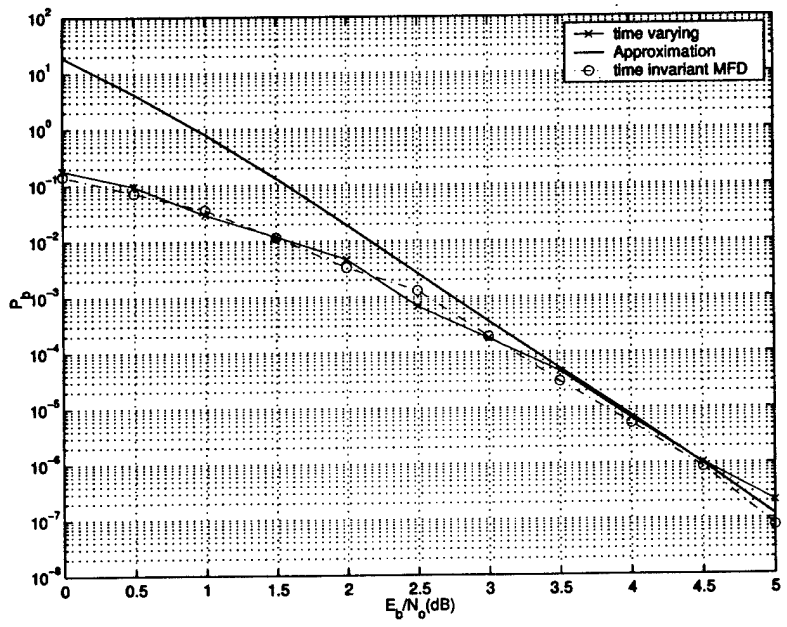

Figure 21: Simulation of (2,1,7,2) No.3.

4 Time-Varying Turbo Codes

4.1 Introduction

Due to their near Shannon limit performance, turbo codes have attracted a great deal of attention since their discovery in 1993 [13]. The BER performance of a turbo code may be divided into two regions. The so-called waterfall region in which the BER drops rapidly and the error-floor region where the BER drops at a slower rate. It is desirable to have turbo codes that have fast-dropping waterfalls and low error-floors when they are decoded with the iterative decoding algorithm. However, it is challenging to design turbo codes that perform well in both regions due to conflicting design constraints.

It is well-known that the error-floor is caused by the relatively small free distance of turbo codes with pseudorandom interleavers [15, 16]. At moderate to high SNRs, a turbo code with a pseudorandom interleaver will perform worse than conventional codes having large free distance. It is intuitive to use large memory convolutional component codes to build turbo codes having large free distances and hence lower error-floors. Unfortunately, the iterative decoding algorithm for turbo codes generally does not converge for component codes with memory greater than 4 [19] and thus the performance in the waterfall region is significantly compromised.

Although challenging, work has been done to design turbo codes that excel in both regions. One example of this is the use of asymmetric turbo codes [17]. That is, turbo codes that have one "weak" component code which performs well in the waterfall region and the other "strong" which gives a large free distance. An asymmetric turbo code consisting of the component code of the Berrou turbo code and a recursive systematic code with a primitive feedback polynomial was constructed in [17]. It turns out that this code is inferior to the Berrou turbo code in the waterfall region, but has a much lower error-floor. The simulated BER performance of this code in shown in Figure 22.

Another encoding scheme that combines "weak" and "strong" together is to use the Big-Numerator Little-Denominator (BN-LD) convolutional codes discovered by Massey et al. [14]. In [14], the turbo code constructed from a memory 8 BN-LD convolutional code with recursive systematic generator matrix

$$G_{BN-LD}(D) = \left[1, \frac{1+D+D^7+D^8}{1+D+D^2}\right]$$

is shown to have slightly better performance than the Berrou turbo code does in both regions. Note that $G_{BN-LD}(D)$ has a large-degree feedforward polynomial and a small-degree feedback polynomial. It is conjectured that this feature enables the turbo code to perform well in all regions. Simulation results of the original Berrou turbo code and the memory 8 BN-LD turbo code are shown in Figure 22. As is indicated by the figure, the memory 8 BN-LD code outperforms the Berrou code at all SNRs.

As an example to illustrate the tradeoff in performance between the two regions, the BER performance of a turbo code based on memory 6 BN-LD component codes with generator matrix

$$G_{BN-LD}(D) = \left[1, \frac{1+D+D^5+D^6}{1+D+D^2}\right]$$

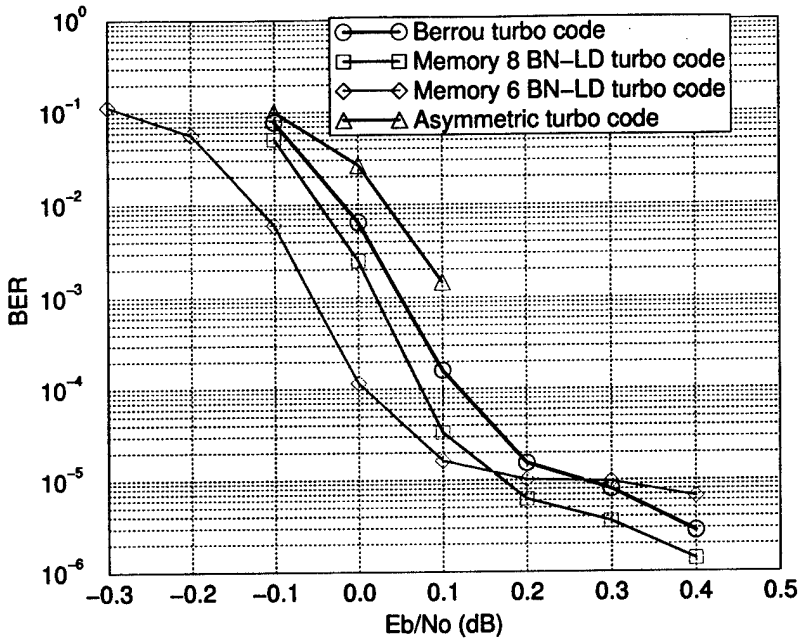

Figure 22: Simulated results of the Berrou turbo code, the memory 8 BN-LD turbo code, and the memory 6 BN-LD turbo code with 18 decoding iterations. All codes are rate 1/3 with random interleaver of length 16384.

is also plotted in Figure 22. Due to its smaller memory, the decoding of the turbo code based on this code is less complex than that of the memory 8 BN-LD turbo code and, since it is "weaker" than the memory 8 BN-LD code, the turbo code constructed is expected to perform better in the waterfall region. The EXIT analysis described in [19] provides a useful tool for predicting the BER performance of turbo codes in the waterfall region. EXIT charts for the component codes of the Berrou code, the memory 8 BN-LD code and the memory 6 BN-LD code are shown in Figures 23, 24, and 25, respectively. The EXIT charts show that at SNRs greater than -0.3 dB the iterative decoding algorithm converges for both the Berrou and the memory 8 BN-LD codes, but the memory 6 BN-LD turbo code converges for SNRs greater than -0.4 dB. As shown in Figure 22, the BER performance of the memory 6 BN-LD turbo code is in fact better in the waterfall region than both the Berrou and memory 8 BN-LD turbo codes, but its error-floor is the worst among the three.

Thus, by using a weaker component code, the performance in the waterfall region improves and the decoding complexity decreases. However, the performance in the error-floor regions is worsened, presumably due to a decreased free distance. The question remains as to whether or not the waterfall performance of the memory 6 BN-LD code can be retained while improving its performance in the error-floor region. In this paper, time-varying component codes will be used to address this.

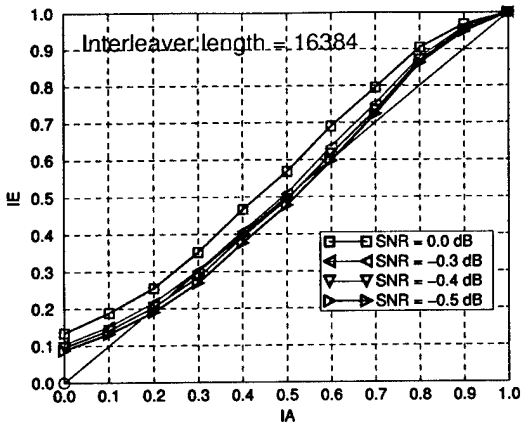

Figure 23: EXIT chart of the Berrou turbo code with interleaver of length 16384.

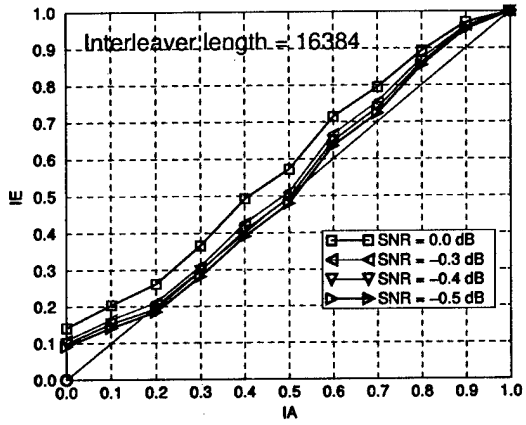

Figure 24: EXIT chart of the memory 8 BN-LD turbo code with interleaver of length 16384.

4.2 Time-varying Component codes

A time-varying convolutional code [20] is a convolutional code generated by a convolutional encoder whose generator matrix changes with time. Denote the generator matrix of a time-varying convolutional encoder at time t as $G_t(D)$. A time-varying convolutional encoder is said to be periodic with period T if $G_t(D) = G_{t+T}(D)$ for all $t = 0, 1, \cdots$. Since time-invariant convolutional codes can be seen as generated by periodic time-varying convolutional encoders with period 1, they are a subset of time-varying convolutional codes. It is conjectured that the use of time-varying component codes, which can consist of combinations of weak and strong time-invariant codes, will enable the construction of turbo codes with better performance in both regions.

The first time-varying component code considered is a memory 6, period 2, rate 1/2 time-varying convolutional code, denoted as PTVCC1, and described in Table 8. Obviously, PTVCC1 is con-

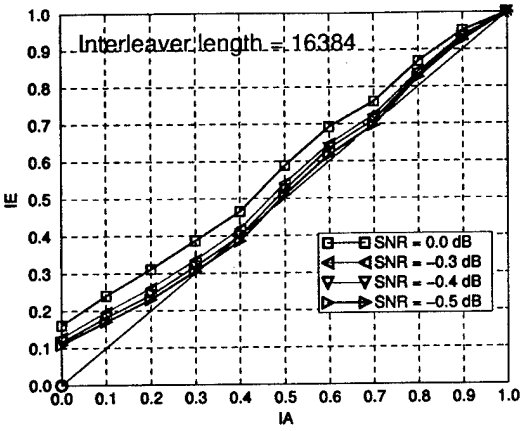

Figure 25: EXIT chart of the memory 6 BN-LD turbo code with interleaver of length 16384.

structed according to the same principles as in [14]. That is, it is a BN-LD code. The EXIT analysis of this code, shown in Figure 26, indicates that the iterative decoder would start to converge for SNRs greater than -0.5 dB and would converge relatively fast for SNRs greater than -0.4 dB. Denote the turbo code based on PTVCC1 with a pseudorandom interleaver as TC1. The simulation results for TC1 are plotted in Figure 27. The Figure shows that the BER performance of TC1 is about 0.1 dB better than that of the memory 8 BN-LD turbo code in the waterfall region which is consistent with the EXIT analysis. It also performs better than the memory 6 BN-LD turbo code in both the waterfall and error-floor regions. However, the simulation also shows that TC1 still has a much higher error-floor, suggesting a smaller free distance, than both the memory 8 BN-LD turbo code and the Berrou turbo code.

	$G_0(D)$	$G_1(D)$
PTVCC1	$\left[1, \frac{1+D+D^5+D^6}{1+D+D^2}\right]$	$\left[1, \frac{1+D^5+D^6}{1+D+D^2}\right]$
PTVCC2	$\left[1, \frac{1+D+D^5+D^6}{1+D+D^2}\right]$	$\left[1, \frac{1+D^2+D^3+D^5+D^6}{1+D+D^2+D^3+D^6}\right]$

Table 8: Generator matrices of the time-varying component codes used to build turbo codes.

In order to improve on the performance of the TC1 in the error-floor region, a second period 2 time-varying component code based on memory 6 convolutional codes is constructed. This code, denoted by TC2, consists of a BN-LD code and a traditional high free distance convolutional code. The parameters of this code is given in Table 8. The idea behind this code is that it would combine the "weak" aspects of the BN-LD code and the strong aspects of the high free distance code. The EXIT analysis for TC2 is shown in Figure 28 and indicates that the iterative decoder will converge for SNRs above -0.4 dB. The performance of the resulting turbo code, denoted by TC2, based on TC2 is shown in Figure 27. As expected from the EXIT analysis the performance in the waterfall region is 0.1 dB worse than TC2. However, the performance in the error-floor region is significantly

improved.

As mentioned in the introduction, the most common method for improving performance in the error-floor region is to use spread interleavers [18]. Figure 29 shows the performance of all five turbo codes considered in this paper with the same spread interleaver with a spreading factor of 20. Though the spread interleaver improves the error-floor of the all the turbo codes, the relative performance remains unchanged. It is also noted that the error floor of both the turbo code with memory 8 BN-LD component code and the turbo code with PTVCC2 as component code disappear, but the turbo code with PTVCC2 as component code achieves this with a lower decoding complexity due to decreased memory.

4.3 Conclusion

It is shown in this paper that good turbo codes can be constructed using time-varying convolutional codes. A turbo code based on a simple period 2, memory 6, time-varying component code was shown to outperform the memory 8 BN-LD code of [14] with less decoding complexity. By combining the weak properties of BN-LD codes with the strong properties of traditional high free distance convolutional codes, time-varying codes offer the possibility of improvement in both the waterfall and error-floor regions. Work is continuing on finding turbo codes based on time-varying component codes that show this.

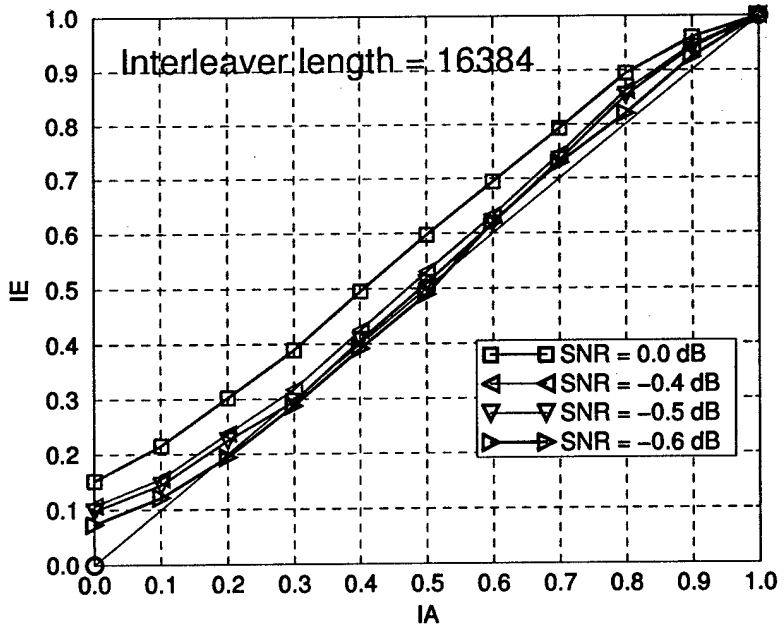

Figure 26: EXIT chart of the TC1 with interleaver of length 16384.

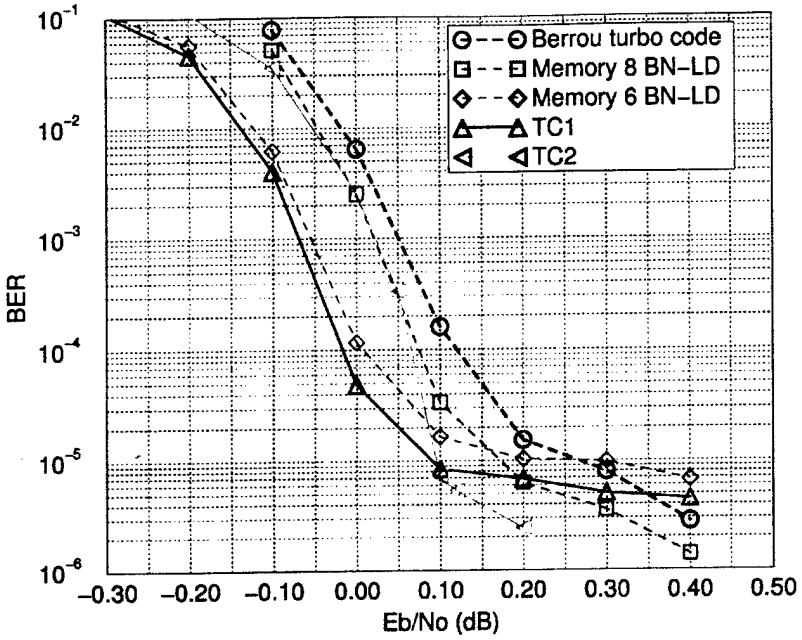

Figure 27: Simulated BER performance of the Berrou turbo code, the memory 8 BN-LD turbo code, the memory 6 BN-LD turbo code, TC1, and TC2 with 18 decoding iterations. All codes are rate 1/3 with random interleaver of length 16384.

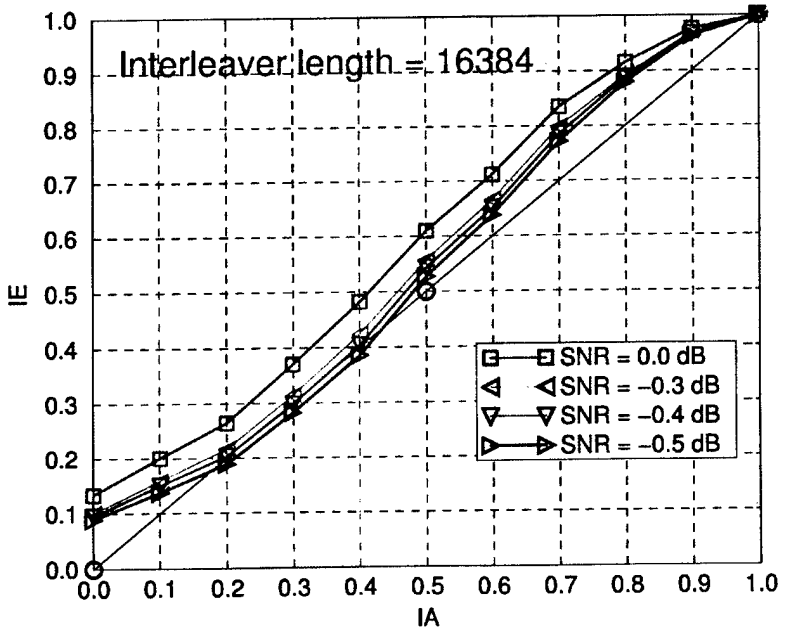

Figure 28: EXIT chart of TC2 with interleaver of length 16384.

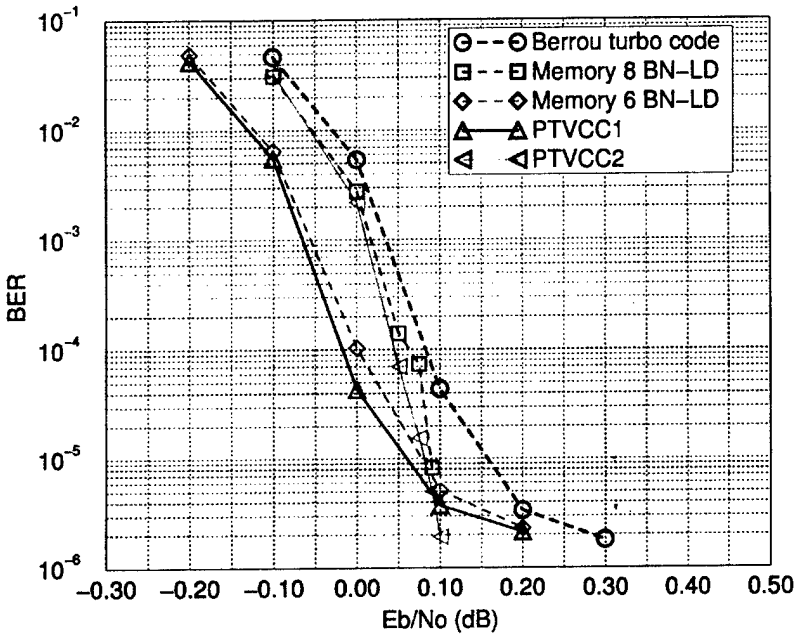

Figure 29: Simulated BER performance of the Berrou turbo code, the memory 8 BN-LD turbo code, the memory 6 BN-LD turbo code, TC1, and TC2 with 18 decoding iterations. All codes are rate 1/3 with spread interleaver of length 16384 and spreading factor 20.

5 Joint Source-Channel Coding

5.1 Introduction

This report aims at comparing different joint source and channel coding techniques in the literature. Since a joint source and channel coding system performs both source coding and channel coding, the performance evaluation of such a system must be based on how well it compresses the data given a distortion constraint as well as how successfully it combats against the channel errors. In this study, 8-bit, monochrome images are taken as the source and are transmitted over a binary symmetric channel (BSC). Therefore, the performance of the system is evaluated using the following three measures:

1. **The overall rate** of the system (taking both the rate reduction due to source coding and rate increase due to channel coding into consideration).
2. **The distortion** after joint source and channel coding. To evaluate the distortion of the images, the signal-to-noise ratio (SNR) measure is used.
3. **Probability of channel error** of the BSC.

Given these three factors, the goal of a joint source and channel coding system is to minimize the distortion (therefore maximize the SNR) for a given rate and probability of channel error, or to minimize the overall rate for a given distortion and channel error probability.

The report is organized as follows: In Section 2, the motivation behind joint source and channel coding is explained from a somewhat information theoretical point of view. In Section 3, a classification for different approaches to joint source and channel coding is made, which covers a brief discussion of the significant contributions to the area. In Section 4, a joint source and channel coding scheme that exploits the residual redundancy of subband coded images is presented. In Section 5, the performance of a wavelet based, progressive coding scheme is investigated for noisy channels. In Section 6, the performance of channel-optimized vector quantization is investigated.

5.2 The Idea of Joint Source and Channel Coding

In classical communication systems, the design of the source coder and channel coder have been made separately. This is due to the fact that the data compression does not depend on the channel and error control coding does not depend on the source distribution [22]. Shannon, in his original paper [21] proved that the two-stage method is as good as any other method of transmitting information over a noisy channel. This result, known as the *separation theorem*, has some important practical implications. It implies that we can consider the design of a communication system as a combination of two parts, source coding and channel coding. We can design source codes for the most efficient representation of the data. We can separately and independently design channel codes appropriate for the channel. The combination will be as efficient as anything we could design by considering both problems together [22].

However, as we try to operate under more and more restrictive conditions, the separation of source and channel coders becomes impractical to implement. It has been shown that the separation does not hold for all channels [23]. Also, there are examples of multiuser channels, which are the channel models for today's wireless communication systems, where the decomposition breaks down. Moreover, even for the channels where the separation holds, the design of an optimal source and channel coder pair is needed which is usually impractical to implement.

Another reason for why the separation theorem does not hold in practice is that it assumes that the source encoder outputs an independent sequence for optimal channel coding. However, since the source encoders of practical interest are not optimal, their outputs contain redundancy. Also, the channel coders are assumed to be optimal in the sense that they produce the source encoder output at the source decoder input with negligible distortion which is an unrealistic assumption due to channel errors that remain uncorrected. Therefore, the non-optimality of source and channel coders causes the separation axiom to breakdown.

All the weaknesses of the separation theorem mentioned above motivated researchers to find more efficient ways of doing source and channel coding. Various approaches to the solution of the problem have been developed and are usually grouped under the general heading of *joint source and channel coding*. The next section briefly summarizes these approaches.

5.3 Background

The joint source and channel coding schemes in the literature can be classified into four categories.

The first class of these schemes is the *joint* source and channel coding schemes, named as such since the source and channel coding operations are truly integrated into one coder structure. The most important examples of this category include the work of Ancheta [24] and Massey [25], the work of Dunham and Gray [26] and Ayanoglu and Gray [27], who investigated the design of joint source and channel *trellis* coders. These studies are rather theoretical works and very hard to implement in practice, if not impossible.

The second class of joint source and channel coding schemes are named as *concatenated* source and channel coding schemes. In this type of coders, known source coders and known channel coders are cascaded and an optimal rate allocation between the source coder and the channel coder is performed for maximum system performance. The work in this category [28]-[32] uses known source coding techniques such as different forms of (i.e., two-dimensional, backward adaptive, embedded, etc.) differential pulse code modulation (DPCM), discrete cosine transform (DCT), and tree encoding, and *concatenates* them with different forms of (i.e., short constraint lengthened, self-orthogonal, punctured, etc.) convolutional codes, and Hamming codes. The important issue here is to find the optimal allocation of the fixed rate between the source coder and the channel coder as well as their cooperation.

In a third class, *unequal error protection* source and channel coders are considered. This type of source and channel coding schemes make use of the fact that channel errors in different bits cause different effects on the final reconstruction. Depending on the source coding scheme, errors

in some of the bits cause more distortion than others. Therefore, the bits can be classified as important and unimportant bits. The main idea behind unequal error protection is to heavily protect important bits at the expense of poor protection of the unimportant bits, resulting in better system performance. Work in this field [33]-[35] includes different source coding schemes such as pulse code modulation (PCM), subband coding as well as different error protection methods.

The last category in the classification of the joint source and channel coding schemes include the *constrained* joint source and channel coders. The source coders in this class are modified to account for the presence of a noisy channel. In other words, the source coders are optimized subject to a noisy channel constraint [36]-[38]. One subset of constrained joint source and channel coders are those coders that make use of the knowledge of source coding properties to combat channel errors. These studies [39]-[48] generally utilize the statistical properties of the source encoder output such as the residual redundancy and try to detect and/or correct channel errors.

5.4 Joint Source and Channel Coding of Subband Coded Images Using Residual Redundancy

In this section, we investigate the performance of a joint source and channel coding system that exploits the residual redundancy at the source encoder output to detect and correct channel errors. The source coder is a concatenation of a subband coder, a DPCM coder and a Huffman coder. For channel coding, a non-binary convolution encoder is proposed which is optionally used to perform error correction. The channel output is decoded using a variable-length list Viterbi decoder and the source decoders. The overall system diagram is shown in Figure 30.

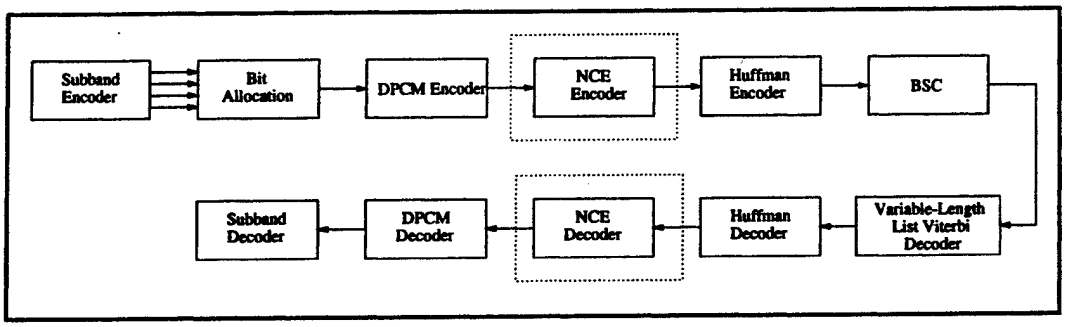

Figure 30: System diagram of a joint source-channel coding scheme.

Natural sources such as images usually have low pass characteristics. Therefore, most of the information in these sources tend to be in the low frequency bands. Subband coding is a source coding scheme that decomposes the source into its subbands, and codes each subband according to its information content. One of the most commonly used ways of looking at the information content of a subband is to calculate its energy. A subband that carries higher energy than other subbands has greater information content. Therefore, a reasonable way of coding the subbands of an image would be to allocate more bits to the subbands with higher energy and allocate fewer bits to the subbands with lower energy. The problem of allocating bits to subbands, namely bit allocation, is one of the challenging problems in source coding and a number of different bit allocation schemes have been proposed in the literature. These schemes aim at minimizing the overall distortion

subject to a given rate constraint.

The subband decomposition for images is often carried out in the following manner: First, the rows of the image are filtered. Usually, two types of decomposition filters are used: low-pass and high-pass. After filtering the rows using low-pass and high-pass filters, we end up with two subbands of the image: low band and high band. Next, the outputs of the filters are subsampled. The justification for the subsampling is the Nyquist rule which states that twice as many samples per second as the range of frequencies suffice for perfect reconstruction. Since after filtering, the range of frequencies for each subband is halved as compared to the original image, we need only half of the samples at the output of each filter. Therefore, a 2:1 subsampling can be done without loss of any information.

After low and high pass filtering of the rows and subsampling, exactly the same operations are performed on the columns for both low and high bands. The two-stage filtering operation on the rows and columns is equivalent to a two-dimensional filtering. After filtering and decimation, four subbands or subimages are obtained. The subimage obtained by low-pass filtering the rows and columns is called the low-low (LL) image. The subimage obtained by low-pass filtering the rows and high-pass filtering the columns is called the low-high (LH) image. The subimage obtained by high-pass filtering the rows and low-pass filtering the columns is called the high-low (HL) image. Finally, the subimage obtained by high-pass filtering the rows and columns is called the high-high (HH) image. Since subsampling is performed after each filtering operation, an image of dimension $N \times N$ results in four subimages of dimensions $\frac{N}{2} \times \frac{N}{2}$. The subband coding procedure has been depicted in Figure 31.

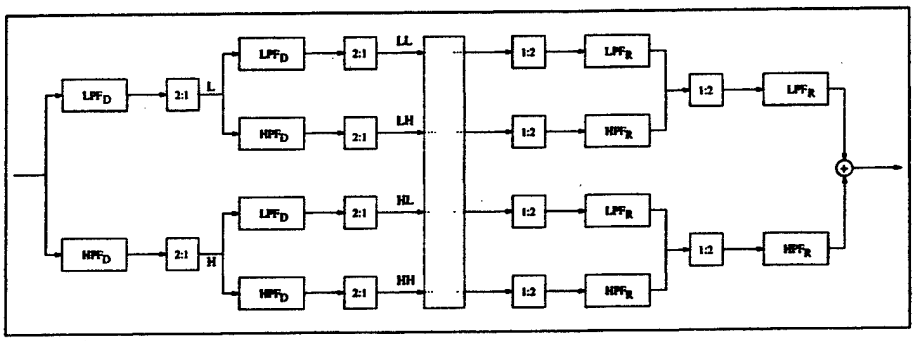

Figure 31: System diagram of a subband coding scheme.

One of the most important design problems in subband coding is the filter implementation. In the literature, there are a number of filter pairs (low-pass and high-pass) proposed for subband coding. These filters are often realized as causal, finite impulse response (FIR) filters. Another type of filter often used in subband coding perform wavelet decomposition in terms of FIR filters.

In the proposed scheme, after the subband decomposition and the decision for allocating the number of bits to each subimage has been made, the subimages are encoded using DPCM which is a differential coding scheme where the difference values to be quantized and coded are minimized using linear prediction. Next, the output of the DPCM encoder is further encoded using Huffman

coding. Huffman coding is a lossless variable-length coding scheme, therefore the output of our concatenated source coder has variable-length codewords.

The test image used throughout the simulations is the Sena image shown in Figure 32. The subband coder uses 9/7-tap FIR filters that employ wavelet decomposition [58]. The image edges have been reflected exploiting the symmetric structure of the filter coefficients to reduce the distortion. The DPCM coder used in this study has a three-bit uniform quantizer and a third order predictor that predicts the value of a subband coefficient using three neighboring coefficients. The predictor coefficients are calculated using the autocorrelation function of the image to minimize the mean square error. This process is a modification of the well-known Wiener-Hopf equations. The DPCM coder is used to code only the LL subimage since the rate allocation schemes used in the study allocates all three bits to the low-low subband. The codebook of the Huffman coder is formed using a training image. Therefore, it is assumed to be known at both the encoder and the decoder.

Figure 32: Test image used for simulation results.

At the output of the channel, a list Viterbi decoder with branch metric that accounts for both the residual structure in the source encoder output as well as the channel probability of error is used. More precisely, if we denote the output of the source encoder by sequence $\{y_i\}$ and the channel output as $\{\hat{y}_i\}$, the branch metric, L is computed as

$$L = \log P(y_i|y_{i-1}) + \log P(\hat{y}_i|y_i). \tag{3}$$

The first term in (3) is the transition probability from symbol y_{i-1} to y_i. If the source encoder was a perfect coder, then it would produce uncorrelated symbols. However, since it is not ideal, there is still some structure in the output sequence. This is often expressed as *residual redundancy*. This redundancy helps the decoder correct channel errors. The second term accounts for the channel effect and is subject to a practical scaling factor. As the channel probability of error gets smaller, the second term dominates the first term which makes sense since in small error probabilities, it is less likely that a bit will be in error. In a noisier channel, however, the first term dominates the second term which basically means that the bits are more likely to be in error. Therefore, the first term that is a measure of the residual redundancy at the source encoder output should prevail the second term. The source statistics that make up the first term in the branch metric is calculated using a training image that has similar statistical characteristics with the image transmitted. The second metric is calculated assuming that the channel is a BSC with known transition probability.

The list Viterbi decoder generates a list of B globally best candidate sequences after a trellis search. Through the trellis, for each node, B branches with minimum costs survive out of the $N \times B$ candidates, where N is the number of states in the trellis. Depending on the length of the list (B) and the number of states (N), the algorithm can be substantially complex in terms of the number of computations required. It should also be noted that, since the source encoder outputs variable length symbols (due to the Huffman encoder), the list Viterbi decoder incorporates them in the trellis in the form of variable length states. Therefore, different paths entering a state use up a different number of bits from the received sequence.

The image to be transmitted is encoded on a row-by-row basis, and the encoded rows are packetized and transmitted over the BSC. The packets also carry the number of bits that they convey. This information is carried in the header and is assumed to be error-free. Among the paths in the list, the path with the smallest cost and correct number of bits is chosen and decoded as the received sequence. If there is no path in the list with correct number of bits, the one with the smallest cost is chosen.

This system has been simulated for a BSC with transition probabilities ranging from $p = 10^{-5}$ to $p = 10^{-1}$. The performance is evaluated using the *peak signal-to-noise ratio* (PSNR) as the quality measure. The PSNR is defined as

$$\text{PSNR} = 10 \log_{10} \frac{\sum u_p^2}{\sum (u_i - \hat{u}_i)^2}, \tag{4}$$

where u_i is the actual pixel value, $\hat{u}_i$ is the reconstructed pixel value, and u_p is the maximum pixel value, which is 255.

The simulation results have been shown in Figure 33 (green curve). The PSNR value of the reconstructed image starts at 34.879 dB and decreases gracefully as the channel gets noisier. The rate of the overall system is 0.463 bpp. It should be noted that the 3-bit DPCM coder encodes only the LL subimage which gives a rate of 0.75 bpp. The Huffman encoder further reduces this rate to 0.463 bpp. Considering the fact that the input to the Huffman encoder is already a source coded data, a lossless compression of ratio $0.75/0.463 = 1.62$ is substantial. The reason for this relatively high compression ratio is due to the uniform structure of the quantizer used in the DPCM coder.

Next, to make the performance of the proposed system more robust to channel errors, some amount of redundancy is added to the source coder. A nonbinary convolutional encoder (NCE) [44] is used for this purpose. The structure of the rate 1/2 NCE is shown in Figure 34.

The input to the NCE, x_n is the output of the 3-bit DPCM encoder, with is selected from the alphabet $\{0, 1, 2, \ldots, N-1\}$, where $N = 2^3 = 8$. As the output of the rate 1/2 NCE is related to the input by the relation $y_n = Nx_{n-1} + x_n$, the output alphabet becomes $\{0, 1, 2, \ldots, M-1\}$, where $M = N^2 = 64$ as follows from the input-output relation given above. This NCE outputs 6 bits for every 3 bits input to it. Therefore, it is a rate 1/2 coder. Given any 6-bit output symbol at time $n-1$, the NCE outputs a *limited* number of symbols from its output alphabet at time n. Specifically, given a value for y_{n-1}, y_n can take on a value from $\{\alpha N, \alpha N+1, \alpha N+2, \ldots, \alpha N+N-1\}$. For example, if at time $n-1$, the output is 19 (010011), the output of the NCE at time n can only be one of the following symbols: $\{24, 25, 26, 27, 28, 29, 30, 31\}$. Notice that while the encoder output alphabet size is of size N^2, at any given instant the encoder can only emit one of N different symbols. This property of the rate 1/2 NCE gives the channel decoder an improved ability to detect

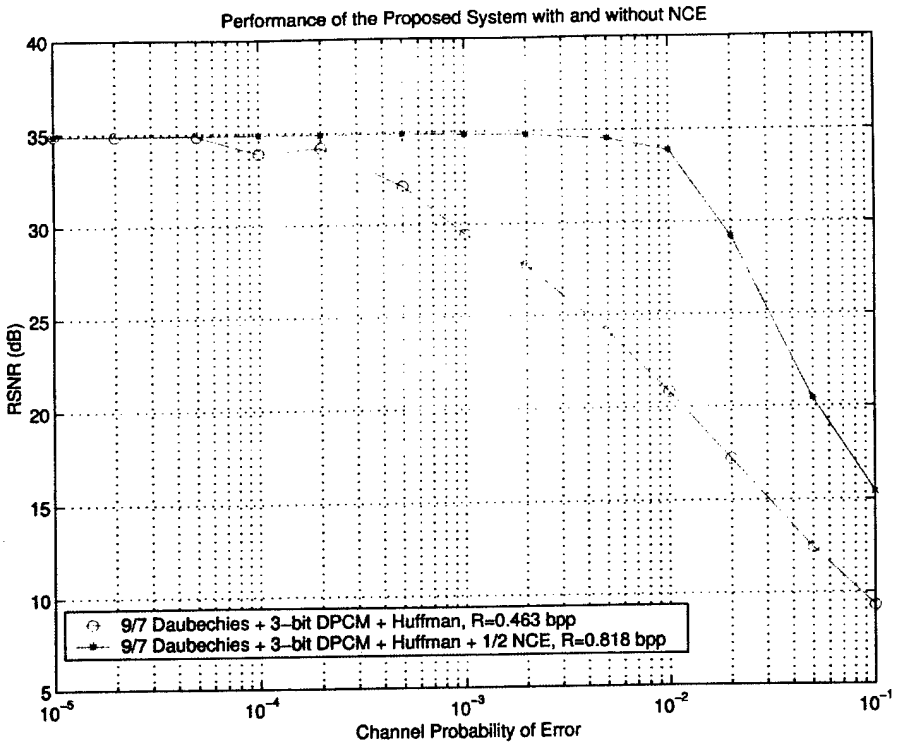

Figure 33: Performance comparison of a concatenated joint source-channel coding schemes with and without a nonbinary convolutional code.

and correct channel errors at the expense of increased rate. It should also be noted that the NCE is designed to ensure that its input alphabet matches the output alphabet of the DPCM encoder. This helps the residual structure in the DPCM output to be maintained for channel coding.

The results for the proposed system with the rate 1/2 NCE is shown in Figure 33 (red curve). It is clear that the PSNR values remain almost constant until $p = 10^{-2}$. The use of the NCE makes the system more robust to channel errors at the cost of increased rate as compared to the system without the NCE. It should be noted, however, that using the NCE increases the number of states in the decoding trellis substantially (depending on the rate) which yields a more complex system in terms of the number of computations required. The rate of the overall system is 0.818 bpp. The rate of the system without the NCE was 0.463 bpp. It might be expected that using a rate 1/2 NCE would double the rate, i.e. increase to $2 \times 0.463 = 0.926$. However, since the Huffman encoder is at the output of the NCE, it helps to reduce this rate to 0.818 bpp, that corresponds to a compression ratio of 1.13.

5.4.1 Comparison of the Joint System to a Separated System

The performance of the previous joint source and channel coding system using the rate 1/2 NCE was compared to the traditional separated system shown in Figure 35. The convolutional code used was the maximum free distance rate 1/2 memory 6 code. The average rate of the separated system

was 0.95 bpp. The simulated channel was the additive white Gaussian noise (AWGN) channel using both hard and soft decisions at the decoders. The channel is assumed to be power constrained so in order to compare the systems with different rates the performances are plotted versus E_p/N_0 where E_p is the energy per pixel. The energy of each transmitted bit, E_b, is determined by the rate of the system, R (bpp) according to $E_b = E_p/R$. Figure 36 shows the simulation results. The simulated performances show that the separated system outperforms the joint system except at small SNR.

5.5 Progressive Image Transmission over Noisy Channels

In this section, we will present one of the novel source coding schemes in the literature, and investigate its performance over noisy channels.

Set Partitioning In Hierarchical Trees (SPIHT) [57] is one of today's most successful and practical image coders for the noiseless channel. It has been shown to outperform almost any other existing source coding scheme. It is computationally simple and has a *progressive* mode of transmission, which means that as more bits are transmitted, better quality reconstructed images can be produced at the receiver. The receiver need not wait for all bits to arrive before decoding, it refines the decoded image with the arrival of each bit of information.

SPIHT is a wavelet-based coding technique, it uses the 9/7-tap FIR filters that were discussed in the previous section. However, the depth of decomposition is different, it decomposes the low-low subimages using the fact that most of an image's energy is concentrated in the low frequency components. We can view the output of a subband coder as in Figure 37.

Since the energy is concentrated in the low frequency components, we can keep decomposing the low-low subimages to perform finer coding on those components. In Figure 38(a), the low-low subimage has further been decomposed to obtain 2-level decomposition. In this case, the decomposition results in 7 subbands. For an image of dimension $N \times N$, the four subbands of the low-low subimage, namely the LLLL, LLLH, LLHL, and LLHH subbands will have dimensions of $\frac{N}{4} \times \frac{N}{4}$ since subsampling is employed after each filtering operation. The remaining low-high, high-low and high-high subbands will have dimensions of $\frac{N}{2} \times \frac{N}{2}$. In Figure 38(b), the image has been decomposed five times, to result in 16 subbands. The four subbands in the highest level of the

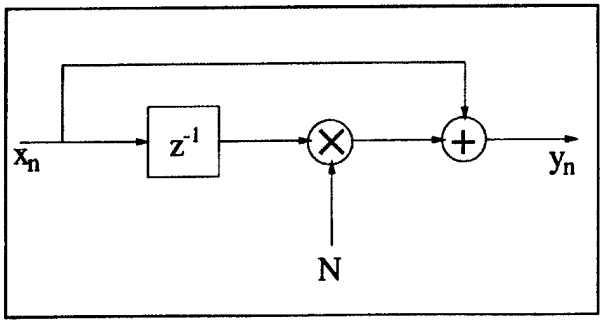

Figure 34: Rate 1/2 Nonbinary convolutional encoder.

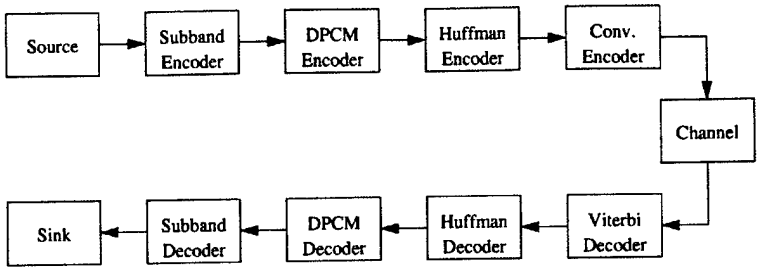

Figure 35: Block diagram of a separated system.

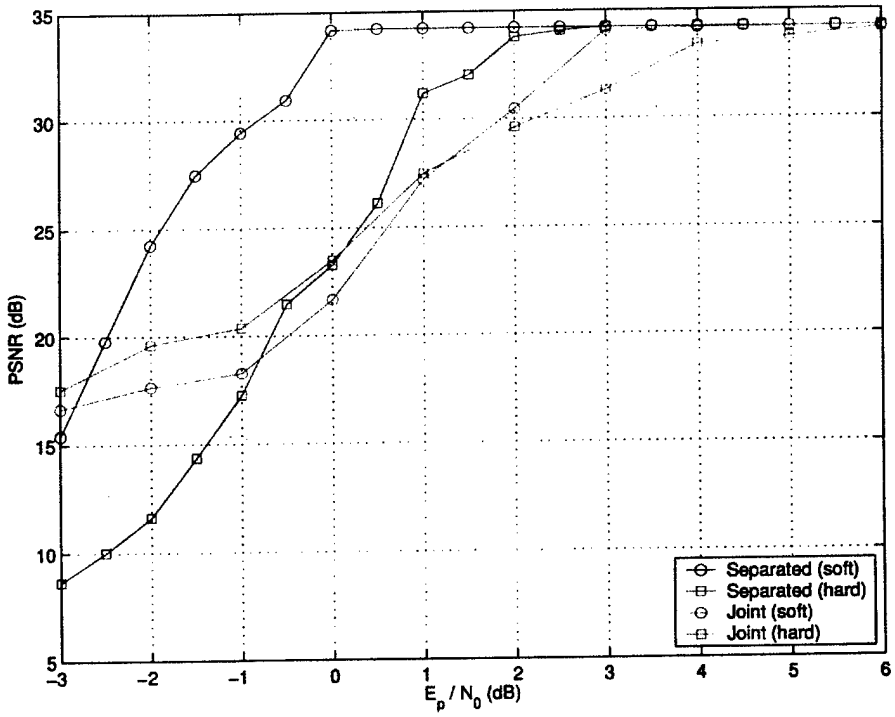

Figure 36: Performance comparison of joint and separated systems.

pyramid have dimensions of $\frac{N}{32} \times \frac{N}{32}$. This form of image decomposition is often referred to as hierarchical subband transformation or pyramid transformation and the lower bands correspond to the higher levels of the pyramid.

To understand the motivation under progressive image transmission, let's define the following notation: Let $p_{i,j}$ be the pixel value of the image at coordinate (i, j). Let $\Omega(\cdot)$ be the unitary hierarchical transformation defined as $\mathbf{c} = \Omega(\mathbf{p})$, where the two-dimensional array $\mathbf{c}$ has the same dimensions of $\mathbf{p}$. $c_{i,j}$ is the transform coefficient at coordinate (i, j). In a progressive transmission scheme, the decoder initially sets the reconstruction vector $\hat{c}$ to zero and updates its components according to the coded message. After receiving the value of some coefficients, the decoder can obtain a reconstructed image using the inverse transform: $\hat{\mathbf{p}} = \Omega^{-1}(\hat{\mathbf{c}})$.

Progressive transmission can be viewed as a way of transmitting the most important information (which yields the largest distortion reduction) first. If we use the mean squared-error (MSE) as the

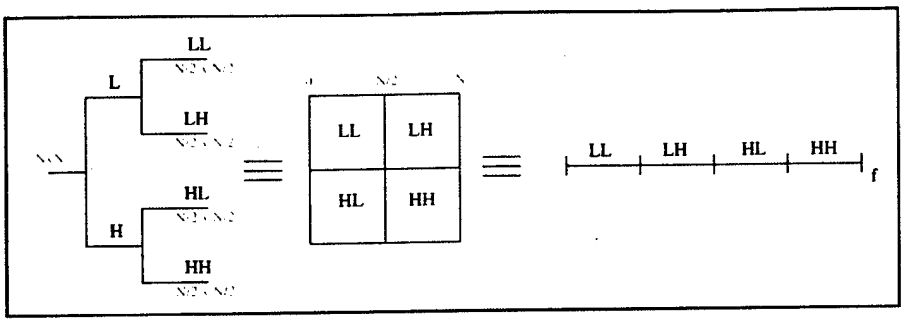

Figure 37: 1-level Decomposition used in SPIHT.

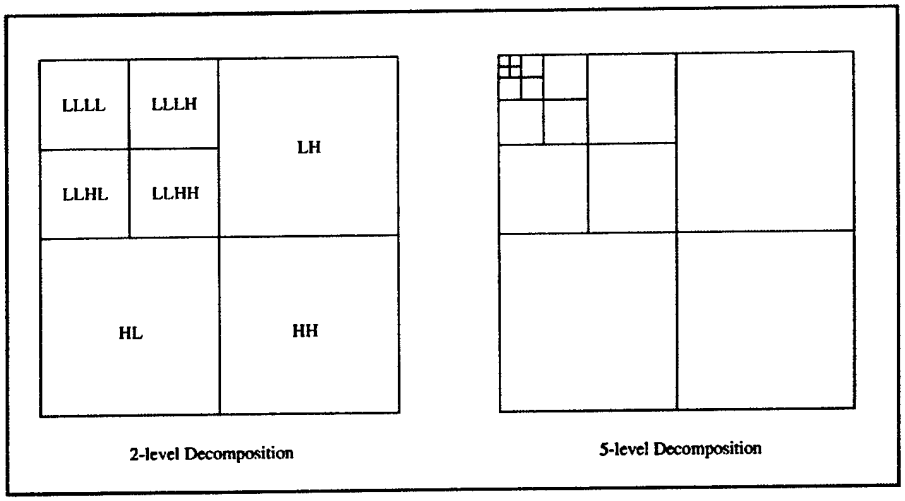

Figure 38: Two level and five level decompositions used in SPIHT.

distortion measure,

$$D_{mse}(\mathbf{p}-\hat{\mathbf{p}}) = \frac{||\mathbf{p}-\hat{\mathbf{p}}||^2}{N} = \frac{1}{N}\sum_i\sum_j (p_{i,j}-\hat{p}_{i,j})^2, \tag{5}$$

we can use the property that the Euclidean norm is invariant to the unitary transformation. Therefore,

$$D_{mse}(\mathbf{p}-\hat{\mathbf{p}}) = D_{mse}(\mathbf{c}-\hat{\mathbf{c}}) = \frac{1}{N}\sum_i\sum_j (c_{i,j}-\hat{c}_{i,j})^2. \tag{6}$$

If the exact value of the transform coefficient $c_{i,j}$ is sent to the decoder, then the MSE decreases by $|c_{i,j}|^2/N$. This means that the coefficients with larger magnitude should be transmitted first because they have a larger content of information. It follows that, the value of $|c_{i,j}|$ can be ranked according to its binary representation, and the most significant bits are transmitted first. In this case, the ordering information should also be transmitted which will increase the rate. However, it is shown [57] that this method of transmitting the information is very efficient despite the fact that a large fraction of the bit-budget is spent in the transmission of ordering information.

The SPIHT algorithm removes the need for the ordering information by *implicitly* transmitting it. It is based on the fact that the execution path of any algorithm is defined by the results of

the comparisons on its branching points. Therefore, if the encoder and the decoder have the same sorting algorithm, then the decoder can *duplicate* the encoder's execution path if it receives the magnitude comparisons, and the ordering information can be recovered from the execution path.

To reduce the number of magnitude comparisons, a *set partitioning rule* is defined that uses an expected ordering in the hierarchy defined by the subband pyramid. The objective is to create new partitions such that subsets expected to be insignificant contain a large number of elements, and subsets expected to be significant contain only one element.

For this purpose the following function is defined:

$$S_n(\Gamma) = \begin{cases} 1, & \max_{(i,j)\in\Gamma}\{|c_{i,j}|\} \geq 2^n \\ 0, & otherwise \end{cases} \tag{7}$$

to indicate the significance of a set of coordinates Γ. The significance of a single pixel value is denoted by $S_n(i,j)$.

The hierarchical relationship between the coefficients in the subband pyramid is defined by a tree structure, called the *spatial orientation tree*. The tree is formed in such a way that each node has either no offspring or four offsprings, which always form a group of 2×2 adjacent coefficients. The coefficients in the highest level of the pyramid are the tree roots and are also grouped in 2×2 adjacent coefficients. However, their offspring branching rule is different, and in each group one of them has no descendants. For a 2-level decomposition (or 2-level pyramid), the structure of the spatial orientation tree is depicted in Figure 39.

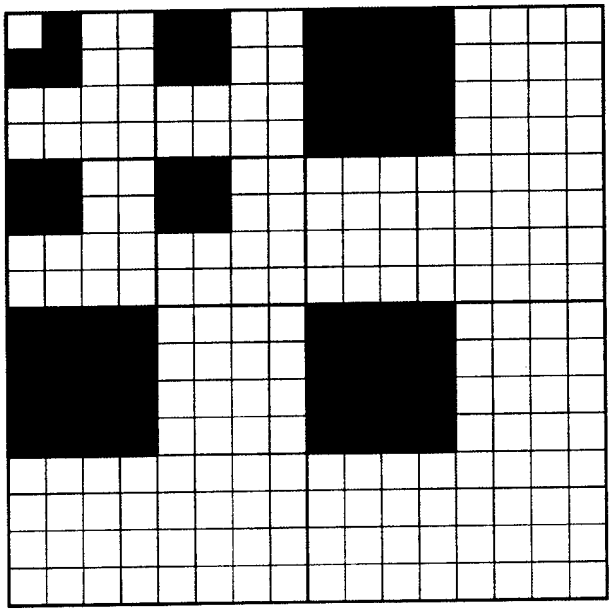

Figure 39: Spatial orientation tree used in SPIHT.

The following sets of coordinates are used to present the new coding method:

- $\mathcal{O}(i,j)$: set of coordinates of all offsprings of node (i,j);

- $\mathcal{D}(i,j)$: set of coordinates of all descendants of the node (i,j);
- $\mathcal{H}$: set of coordinates of all spatial orientation tree roots (nodes in the highest pyramid level);
- $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$

In order to *implicitly* transmit the ordering information, a set partitioning rule is defined according to the following principles:

1. the initial partition is formed with the sets $\{(i,j)\}$ and $\mathcal{D}(i,j)$ for all $(i,j) \in \mathcal{H}$;
2. if $\mathcal{D}(i,j)$ is significant then it is partitioned into $\mathcal{L}(i,j)$ plus the four single-element sets with $(k,l) \in \mathcal{O}(i,j)$.
3. if $\mathcal{L}(i,j)$ is significant then it is partitioned into the four sets $\mathcal{D}(k,l)$, with $(k,l) \in \mathcal{O}(i,j)$.

The SPIHT algorithm works according to the significance of the sets that it partitions as well as the significance of single coordinates. For this purpose, three lists are used: *list of insignificant sets* (LIS), *list of insignificant pixels* (LIP), and *list of significant pixels* (LSP). LIP and LSP store individual pixel coordinates (i,j), on the other hand the LIS stores the sets $\mathcal{D}(i,j)$ or $\mathcal{L}(i,j)$.

The SPIHT algorithm, using the concepts given above, is as follows:

1. **Initialization:** output $n = \lfloor \log_2(\max_{(i,j)}\{|c_{i,j}|\}) \rfloor$; set the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type $\mathcal{D}$ entries.
2. **Sorting pass:**
   (a) for each entry (i,j) in the LIP do:
      i. output $S_n(i,j)$
      ii. if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;
   (b) for each entry (i,j) in the LIS do:
      i. if the entry is of type $\mathcal{D}$ then
         - output $S_n(\mathcal{D}(i,j))$;
         - if $S_n(\mathcal{D}(i,j)) = 1$ then
           - for each $(k,l) \in \mathcal{O}(i,j)$ do:
             * output $S_n(k,l)$;
             * if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
             * if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
      ii. if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type $\mathcal{L}$; else, remove entry (i,j) from the LIS;
   (c) if the entry is of type $\mathcal{L}$ then
      - output $S_n(\mathcal{L}(i,j))$;

- if $S_n(\mathcal{L}(i,j)) = 1$ then
  - add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type $\mathcal{D}$;
  - remove (i,j) from the LIS

3. **Refinement pass:** for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the n-th most significant bit of $|c_{i,j}|$.

4. **Quantization-step update:** decrement n by 1 and go to Step **2**.

The decoding operation is almost exactly the same as the encoding operation. The decoder duplicates the encoder's execution path as it sorts the significant coefficients. The three lists (LIS, LIP and LSP) are the same at both the encoder and the decoder at the same pass, which means that the decoder implicitly recovers the ordering from the execution path, without the need for explicit ordering information. For the value of n when a coordinate is moved to the LSP, it is known that $2^n \leq |c_{i,j}| < 2^{n+1}$. So the decoder uses that information, plus the sign bit that is input just after the insertion in the LSP, to set $\hat{c}_{i,j} = \pm 1.5 \times 2^n$. Also, during the refinement pass, the decoder adds or subtracts 2^{n-1} to $\hat{c}_{i,j}$ when it inputs the bits of the binary representation of $|c_{i,j}|$. In this manner, the distortion gradually decreases during both the sorting and refinement passes with the receipt of each bit, which ensures a perfect progressive mode of transmission.

The SPIHT algorithm has been used for source coding of our test image. For a noiseless channel, the rate vs. distortion performance shown if Figure 40 has been obtained:

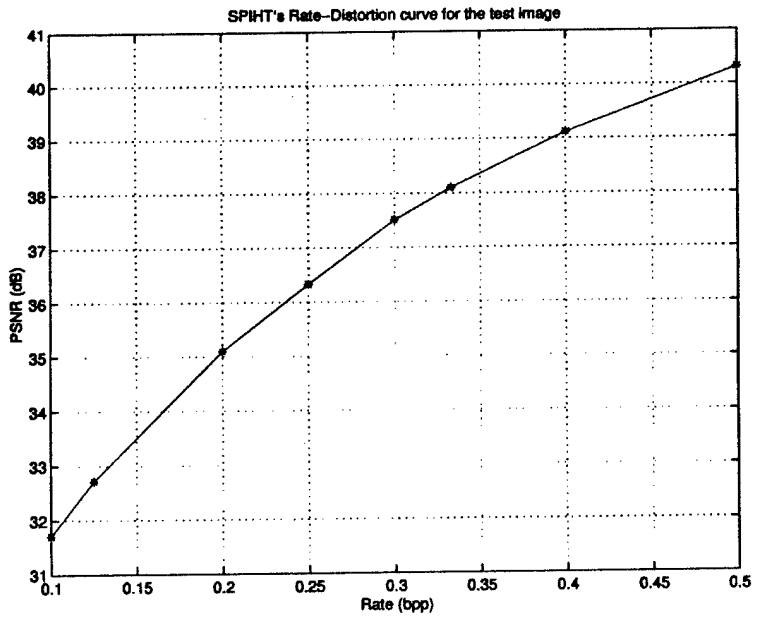

Figure 40: SPIHT's rate versus PSNR performance for the test image.

It is evident that the rate vs. distortion performance of SPIHT is substantially high. It gives relatively high PSNR values even for low rates. For 0.5 bpp, for example, a PSNR value of 40.30 dB is obtained which provides a sufficiently high image quality. To illustrate the high performance of SPIHT, it is compared to vector quantization (VQ). The LBG algorithm [49] is used for vector

quantizer design, and the splitting algorithm is used for initialization. Table 9 depicts the comparison results. The training image used to form the codebook is statistically similar to the test image.

Table 9: Comparison of SPIHT versus VQ as a function of rate.

Rate (bpp)	PSNR (dB)	
	VQ	SPIHT
0.125	24.71	32.71
0.25	28.26	36.32
0.5	32.18	40.30

From Table 9, it is evident that SPIHT outperforms VQ by about 8 dB which is a substantially high difference. For 0.25 and 0.125 bpp, the reconstructed images using VQ and SPIHT are respectively shown in Figure 41. The images in the first column are coded using VQ, and the images in the second column are coded using SPIHT. The first row of images have rate of 0.25 bpp, where the images in the second row have rate of 0.125 bpp.

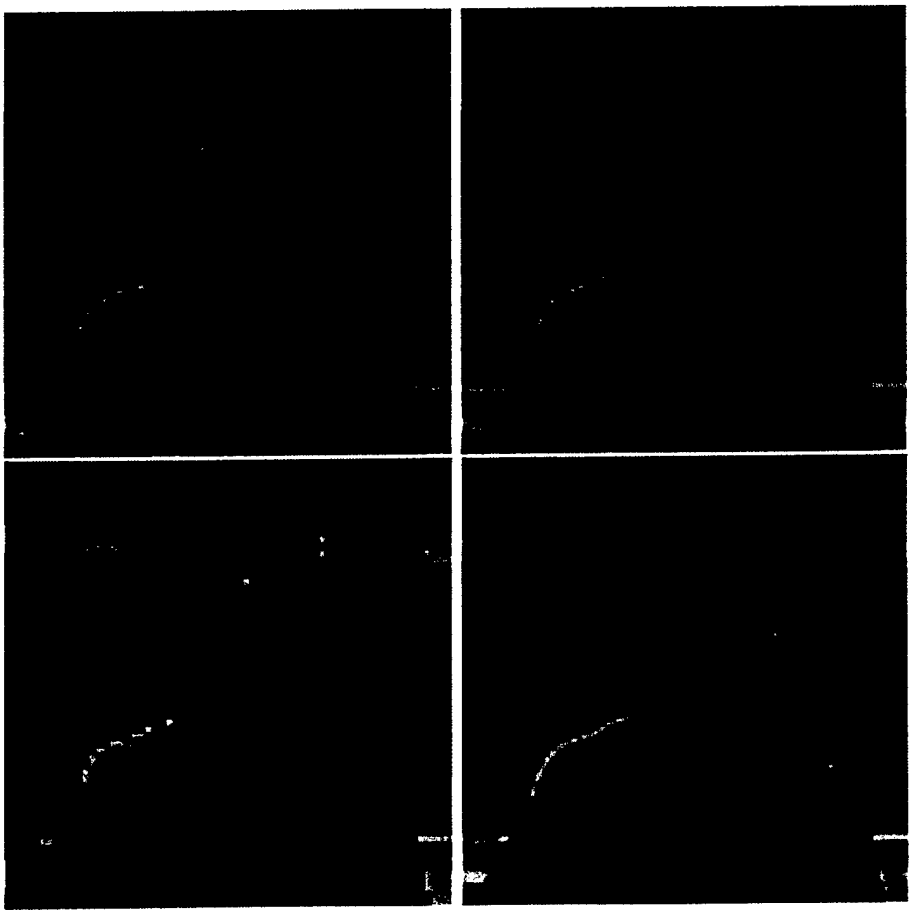

Figure 41: Reconstructed images using VQ (first column) and SPIHT (second column).

While SPIHT has very high performance in noiseless channels, it is very sensitive to channel errors in noisy channels. The channel errors can cause unrecoverable loss of synchronization between the encoder and the decoder. Total collapse of the reconstructed image often results from loss of synchronization. In fact, the vast majority of images transmitted using this progressive wavelet-based algorithm will frequently collapse even if a single transmitted information bit is incorrectly decoded at the receiver.

SPIHT's high source coding performance has led many researchers to find efficient channel coding schemes to reduce its excessive vulnerability to channel errors. The work of Sherwood and Zeger [59] is one of the most successful studies to make SPIHT robust to noisy channels. They partition the output bit stream of the SPIHT coder into consecutive blocks of length N, where they take $N = 200$. Next, they add c ($c = 16$) checksum bits based on the N bits along with zero bits to flush the memory units of the convolutional encoder. The convolutional coder used in this study is a rate-compatible punctured convolutional (RCPC) coder [60]. At the receiver, a list Viterbi decoder with 100 paths is used to choose the path with the lowest path metric that also satisfies the checksum equations.

In our study, we also use the RCPC encoder to channel encode the SPIHT's output bit stream. The RCPC coder is implemented using a $(3, 1, 2)$ convolutional encoder with transfer function matrix $G(D) = [1+D, 1+D^2, 1+D+D^2]$. The critical issue in this joint source and channel coder design is the rate allocation problem. For a given channel probability of error and a given overall rate, the optimum source rate and channel rate should be determined for maximum PSNR performance. Once it is determined, a convenient puncturing rule is employed. The optimum rate allocation for the system has been found using simulations. To make a performance comparison with the scheme explained in the precious section, we have fixed the overall rate to 0.463 bpp which is the same as the overall rate in the previous scheme. And we distributed this rate between the source coder and channel coder.

Simulations have shown that

- for $p \leq 10^{-3}$, a rate 4/6 RCPC encoder with puncturing rule a_1
- for $p = 2 \times 10^{-3}$, a rate 4/7 RCPC encoder with puncturing rule a_2
- for $p = 5 \times 10^{-3}$, a rate 4/9 RCPC encoder with puncturing rule a_3
- for $p = 10^{-2}$, a rate 4/10 RCPC encoder with puncturing rule a_4
- for $p \geq 10^{-2}$, a rate 4/12 (which is the maximum rate for this coder) RCPC encoder with puncturing rule a_5

are suitable choices. For these rates, the corresponding source coding rates and the puncturing rules have been shown in Table 10 and the puncturing rule matrices are below.

$$a_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Table 10: Rate allocation for SPIHT + RCPC System.

Rate Allocation Scheme	Source Coding Rate	Channel Coding Rate and Puncturing Rule	Overall Rate
Scheme I	$R_s = 0.463/1.5 = 0.309$ bpp	4/6 PCC, a_1	$R_t = 0.463$ bpp
Scheme II	$R_s = 0.463/1.75 = 0.265$ bpp	4/7 PCC, a_2	$R_t = 0.463$ bpp
Scheme III	$R_s = 0.463/2.25 = 0.206$ bpp	4/9 PCC, a_3	$R_t = 0.463$ bpp
Scheme IV	$R_s = 0.463/2.5 = 0.185$ bpp	4/10 PCC, a_4	$R_t = 0.463$ bpp
Scheme V	$R_s = 0.463/3 = 0.154$ bpp	4/12 PCC, a_5	$R_t = 0.463$ bpp

$$a_2 = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$a_3 = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$a_4 = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$a_5 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

The performance of the proposed system has been shown in Figure 42. We used the same two rates that were used in the previous section, namely R=0.463 bpp and R=0.818 bpp. In Figure 43 and 44, the performance of this system and the system proposed in the previous section are compared for the same rate pairs. Note that using SPIHT for the source coder and RCPC codes as the channel coder, we can achieve any desired rate. In other words, exact rate control is possible, which is not true for the previous system. It is clear that this new system has superior performance as compared to the previous joint source channel coding scheme. It owes its higher performance to the efficient source coding scheme, namely SPIHT. It should be noted, on the other hand, that an efficient data compression system removes, to the extent possible, the redundancy in the source and retains the useful (nonredundant) part in an effort to reduce the rate. This removal of redundancy, in turn, can introduce a great deal of sensitivity to the channel errors. The channel errors, if not dealt with properly, can result in significant degradations in the performance of the compression scheme. It has been observed that SPIHT, as a very efficient source coding scheme, is also very sensitive to channel errors. It is so vulnerable that total collapse of the reconstructed image is

possible even if a single transmitted bit is incorrectly decoded at the receiver. The channel coder on the other hand is very simple as compared to previous method's variable length list Viterbi decoder with MAP metric. The channel coder can be further improved as in Sherwood and Zeger's [59] work for better error correcting performance.

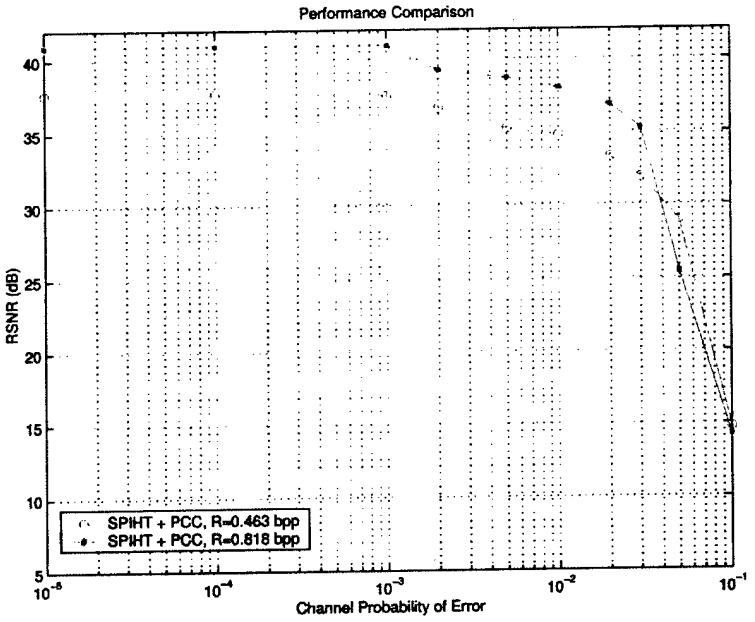

Figure 42: Performance comparison of SPIHT with punctured convolutional coding at different rates.

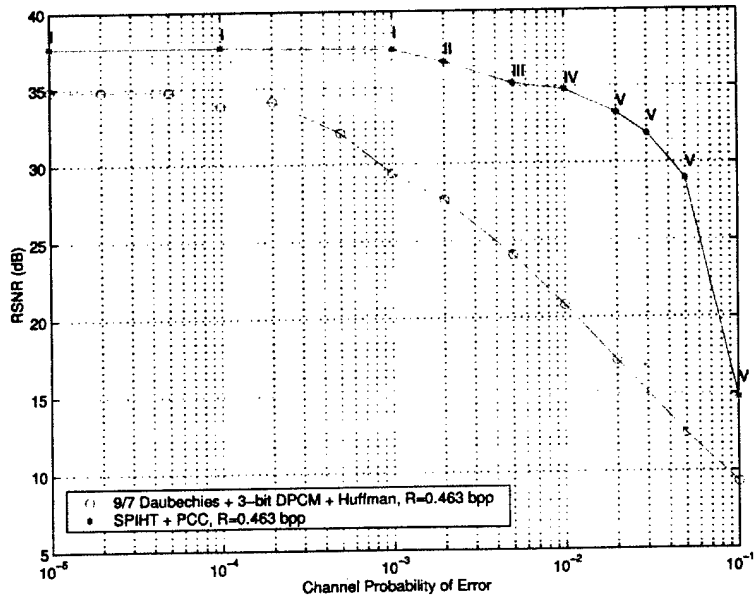

Figure 43: Performance comparison of SPIHT with punctured convolutional coding to a concatenated subband joint-source coding channel scheme.

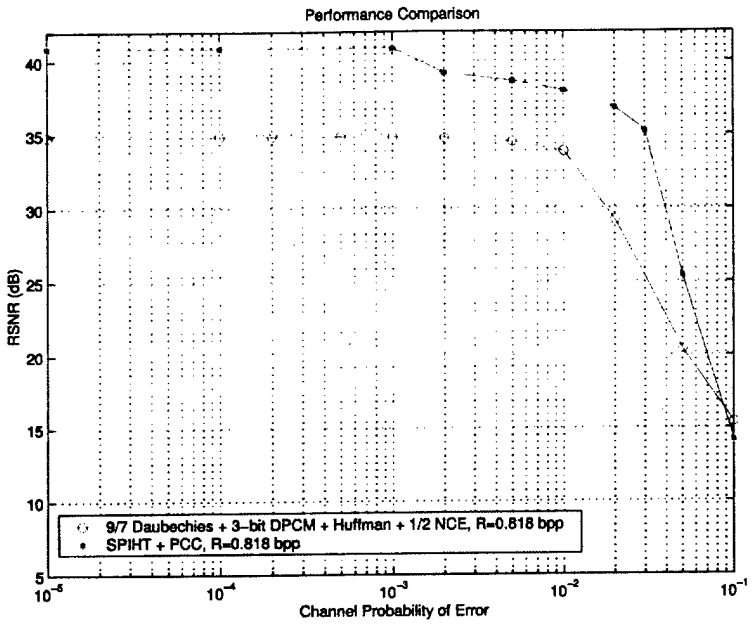

Figure 44: Performance comparison of SPIHT with punctured convolutional coding to a concatenated subband joint-source channel coding scheme with a nonbinary convolutional encoder..

5.6 Channel-Optimized Vector Quantization

Vector quantization (VQ) [49], as a means of data compression, has received a tremendous amount of attention in the past decade. Utilizing vector quantization instead of scalar quantization has resulted in dramatic performance improvements (in terms of rate reduction for a given level of distortion, or reduction of distortion for a given rate) in various image coding applications. The superiority of VQ is due to Shannon's [21] source coding theorem which basically says that encoding sequences of outputs can provide an advantage over the encoding of individual samples. This result, proved by taking longer and longer sequences of inputs, indicates that a quantization strategy that works with sequences or blocks of output would provide some improvement in performance over scalar quantization. This fact is the main motivation behind vector quantization.

In vector quantization, the source output is grouped into blocks or vectors. In image coding, for example, a vector is composed of k pixels. These pixels are chosen to be neighboring pixels in the two-dimensional image plane. This vector of source outputs forms the input to the vector quantizer. At both the VQ encoder and decoder, there is a set of k-dimensional vectors which is called the codebook of the vector quantizer. The codebooks at the encoder and the decoder are the same, and contain M k-dimensional vectors, called the code-vectors.

The operation of a vector quantizer is briefly as follows: The encoder forms the k-dimensional vectors, and compares each vector to each of the M code-vectors in the codebook. For a given input vector, it finds the code-vector (amongst the M possible code-vectors in the codebook) that is closest to the input vector. Next, it transmits the *index* of the code-vector it matches with the input vector. For a codebook of size M, the index is a $\log_2 M$-bit number. The encoder and the

decoder uses these $\log_2 M$ bits to identify a vector of k pixels. In other words, the number of bits per pixel for VQ with codebook size of M and vector size of k is $\frac{\log_2 M}{k}$. This is the rate of this vector quantizer. Finally, the decoder receives the index values. Since it has the same codebook as the encoder, it matches the index values with the corresponding code-vectors, and reconstructs the image using them.

One of the main advantages of vector quantization is its simplicity. The encoder simply makes comparisons with the input vector and M code-vectors in the codebook, and transmits the index of the closest code-vector. The operation of the decoder is indeed simpler: it receives the index and finds the corresponding code-vector using a lookup table.

The main challenge in the implementation of vector quantizers is the codebook design. The determination of the code-vectors that would yield the minimum distortion for a given rate is a nontrivial problem. Various approaches to the design of codebooks for vector quantizers have been proposed in the literature. An efficient, as well as practical method of codebook design for VQ is the Linde-Buzo-Gray (LBG) algorithm [49] that has been widely used in VQ applications. The LBG algorithm is an iterative method of designing VQ codebooks, where at each iteration the distortion is reduced gradually. The algorithm operates on a training set of vectors $\{X_n\}_{n=1}^{N}$, where the training set should have similar characteristics to the source to be coded. The codebook is formed using the training set. Therefore, it is important that the source and the training set should be statistically alike for better coding performance. The LBG algorithm recursively finds and updates quantization regions, $\{V_i^{(k)}\}_{i=1}^{M}$ and the code-vectors (called reconstruction vectors), denoted by $\{Y_i^{(k)}\}_{i=1}^{M}$ at the k-th step. The overall distortion at the k-th step, $D^{(k)}$ is evaluated between the training vectors and the corresponding reconstruction vectors. The recursion and update of the quantization regions and the reconstruction vectors is continued until the average distortion is reduced below the threshold, ϵ. The algorithm, using the notation above, is summarized as follows:

1. Set $k = 0$. Start with an initial set of reconstruction vectors $\{Y_i^{(0)}\}_{i=1}^{M}$ and a set of training vectors $\{X_n\}_{n=1}^{N}$. Set $D^{(0)} = 0$. Choose a threshold ϵ.

2. Update the quantization regions

$$V_i^{(k)} = \{X_n : d(X_n, Y_i) < d(X_n, Y_j) \qquad \forall j \neq i\} \qquad i = 1, 2, \ldots, M. \tag{8}$$

3. Compute the average distortion $D^{(k)}$ between the training vectors and the corresponding reconstruction vectors.

4. If $\frac{D^{(k)} - D^{(k-1)}}{D^{(k)}} < \epsilon$ stop; otherwise, continue.

5. $k = k + 1$. Update the reconstruction vectors $\{Y_i^{(k)}\}_{i=1}^{M}$ that are the average values of the elements of each of the quantization regions V_i^{k-1}. Go to Step 2.

There are some problems in the implementation of the LBG algorithm. One arises in the first step of the algorithm. The performance of the LBG algorithm is closely related with the initial choice of the reconstruction vectors. Different initial conditions selected for the reconstruction vectors yield different distortion values at each step. Therefore, the problem of initialization of reconstruction vectors has a major impact on the performance of VQ. One of the most commonly used ways of

initializing the LBG algorithm is the splitting technique [49]. In this technique, the LBG algorithm starts with a single reconstruction vector (which is the average value of the all training vectors). And at each step, the number of reconstruction vectors is doubled by adding to each of them a perturbation vector. With the doubling of reconstruction vectors, the quantization regions and reconstruction vectors are updated using the LBG algorithm. The splitting technique is a practical and efficient way of initializing the LBG algorithm.

Another problem in the implementation of the LBG algorithm comes into play in the second step of the algorithm. It is possible that, after updating the quantization regions, one (or more) of the new quantization regions may be empty. One way of getting around this problem is to remove the reconstruction vector with no training vector associated with it, and to assign one of the training vectors that belongs to the reconstruction vector with the greatest number of training vectors as the new reconstruction vector.

For an extensive discussion of these and other problems in vector quantization, [50] is a comprehensive reference.

While vector quantization is an efficient source coding technique (as compared to scalar quantization), its performance may be expected to be poorer than that of scalar quantization in noisy channels. This may be justified as follows: a source coding technique becomes more efficient as it reduces more redundancy in the source. As the redundancy is removed, the remaining nonredundant part of the source (source encoded data) becomes more sensitive to the channel errors. SPIHT's, as a very efficient source coding technique, excessive vulnerability to transmission errors is an example of this phenomenon. Similar discussion for the sensitivity of efficient source coding schemes to bit errors was made in the previous section.

In an effort to increase VQ's performance in noisy channels, a number of techniques have been proposed in the literature [51] - [56]. In this section, we implement the *channel-optimized vector quantizer* proposed in [54].

The objective in the design of the channel-optimized vector quantizer is to introduce the effect of the channel in the VQ design process in order to maximize the performance of the vector quantizer for noisy channels. For this purpose, the distortion measure is modified to account for the channel error probability. The design of a channel-optimized vector quantizer is briefly explained below.

A k-dimensional, M-level VQ is designed for a discrete memoryless channel (DMC) with input and output alphabets $\{1, 2, \ldots M\}$. $P(j|i)$ is the probability that index j is received given that index i is transmitted. The k-dimensional source output vector is denoted as $x = (x_{nk}, x_{nk+1}, \cdots, x_{nk+k-1})$ and has a k-fold probability density function (p.d.f), $p(x)$.

The encoder is described in terms of a partition of quantization regions $\mathcal{P} = \{S_1, S_2, \ldots, S_M\}$ according to the following encoder mapping:

$$\gamma(x) = i, \qquad \text{if} \quad x \in S_i, \qquad i \in \{1, 2, \ldots M\}. \tag{9}$$

The decoder is described in terms of the codebook $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$ according to the following decoder mapping:

$$g(j) = c_j, \qquad j \in \{1, 2, \ldots M\}. \tag{10}$$

Let the distortion caused by representing the source vector x by a code-vector y be denoted by $d(x, y)$. The average distortion, D, of this system is

$$D = \frac{1}{k} \sum_{i=1}^{M} \sum_{j=1}^{M} P(j|i) \int_{S_i} p(x) d(x, c_j) dx. \tag{11}$$

Note that the expression of the average distortion given above accounts for the distortion that arises from the source encoder as well as the channel.

The goal in channel-optimized vector quantization (COVQ) design is to choose the codebook $\mathcal{C}$ and quantization regions $\mathcal{P}$ to minimize the average distortion D. It has been shown in [54] that the optimal quantization regions as well as the optimal codebook for the mean squared error distortion measure is evaluated as:

$$S_i^* = \left\{ x : \sum_{j=1}^{M} P(j|i) ||x - c_j||^2 \leq \sum_{j=1}^{M} P(j|l) ||x - c_j||^2, \quad \forall l \right\}, \quad i \in \{1, 2, \ldots M\} \tag{12}$$

$$c_j^* = \frac{\sum_{i=1}^{M} P(j|i) \int_{S_i} x p(x) dx}{\sum_{i=1}^{M} P(j|i) \int_{S_i} p(x) dx}, \quad j \in \{1, 2, \ldots M\}. \tag{13}$$

The equations (12) and (13) are applied recursively: In (12), the quantization regions are evaluated. These quantization regions are then applied in (13) to find the code-vectors which are again invoked in (12) to find the quantization regions. It should be noted that both equations have the term $P(j|i)$ that introduces the channel effect into the design of the COVQ. In this sense, the channel-optimized vector quantizer falls into the category of *constrained* joint source and channel coders, where the source coders are modified to account for the presence of a noisy channel. In COVQ, the vector quantizer is optimized subject to a noisy channel constraint.

The channel-optimized vector quantizer as well as the regular vector quantizer have been simulated for the test image in Figure 32. Different codebooks have been obtained for different channel error probabilities using COVQ. The channel is a BSC and the training image is different from the test image. Codebook size, M, is 16, and the vectors are of dimension 2×2. Thus, the rate is 1 bpp. The results are depicted in Figure 45.

One observation that can be made from the figure is that the COVQ performs slightly poorer than VQ for very low channel error probabilities. As the channel error rate increases, the COVQ performs better. For $p = 10^{-3}$, the COVQ and VQ have almost the same PSNR performance. For higher error probabilities, the COVQ surpasses VQ. For $p = 0.02$, for example, the COVQ outperforms the VQ by about 1.8 dB.

As the codebook size increases, the codebook design of COVQ becomes computationally burdensome. Also, since the number of code vectors increases, code vectors get closer to each other in the Euclidean sense. Therefore, the COVQ does not perform significantly better than VQ. These are the observations from the simulations for $M = 256$ and vector size of 4×4 which yields a rate of 0.5 bpp.

Next, the performance of the COVQ has been investigated for the case where the encoder and the decoder have a wrong estimate of the channel error probability. This is called *channel mismatch*. Let

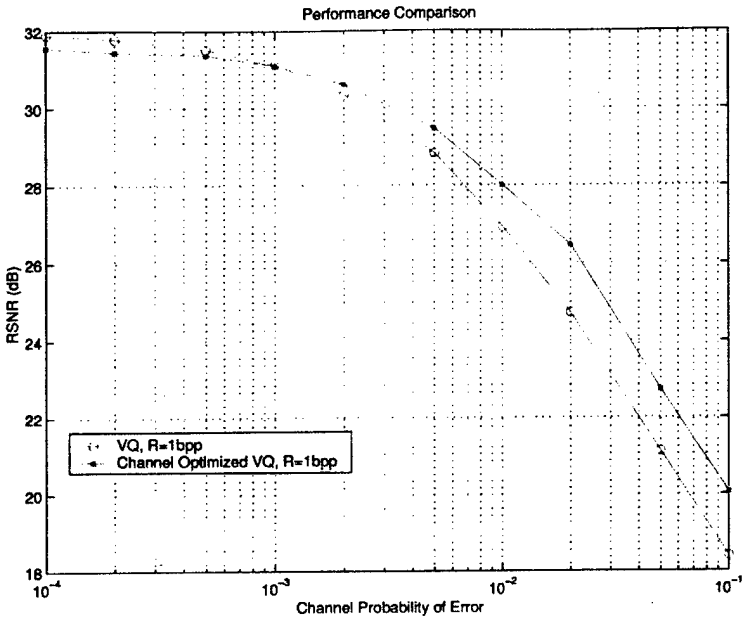

Figure 45: Performance comparison of a vector quantization (VQ) scheme to a channel optimized vector quantization scheme (COVQ).

p denote the actual channel error probability and $\hat{p}$ denote the estimated channel error probability. For $p = 10^{-2}$ and $\hat{p} = 10^{-1}$, which is an overestimation by a factor of 10, the PSNR value is obtained as 26.501 dB which is a drop of 1.51 dB. This value is even worse than regular VQ by 0.488 dB. On the other hand for the same channel error probability, if $\hat{p} = 10^{-3}$ which is an underestimation by a factor of 10, the PSNR is 27.854 dB which is a drop of only 0.157 dB. Moreover, this value is still higher than that of VQ by 0.915 dB. Therefore, the penalty of overestimation of the channel error probability is significantly higher than that of underestimation.

References

[1] M. Mooser, "Some periodic convolutional codes better than any fixed code," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 750-751, Sept. 1983.

[2] R. Palazzo, Jr., "Time varying convolutional encoders better than the best time invariant encoders," *IEEE Trans. Inform. Theory*, vol. 39, No. 3, pp. 1109-1110, May 1993.

[3] P. Joong Lee, "There are many good periodically time-varying convolutional codes," *IEEE Trans. Inform. Theory*, vol. 35, No. 2, pp. 460-463, March 1989.

[4] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, 1999.

[5] J. P. Odenwalder, *Optimal decoding of convolutional codes*, Ph.D. dissertation, Dept. Syst. Sci. Eng. Appl. Sci., Univ. California, Los Angeles, 1970.

[6] K. J. Larsen, "Short convolutional codes with maximal free distance for rate 1/2, 1/3, and 1/4," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 371-372, May 1973.

[7] R. Johannesson, "Robustly optimal rate one-half binary convolutional codes," *IEEE Trans. Commun. Tech.*, vol. COM-21, pp. 464-468, July 1975.

[8] R. Johannesson, "Some rate 1/3 and 1/4 binary convolutional codes with an optimal distance profile," *IEEE Trans. Inform. Theory*, vol. IT-23, pp. 281-183, Mar. 1977.

[9] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New short constraint length convolutional code constructions for selected rational rates," *IEEE Trans. Inform. Theory*, vol. IT-28, No. 5, pp. 794-800, Sept. 1982.

[10] M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," *IEEE Trans. Inform. Theory*, vol. 35, No. 6, pp. 1146-1159, Nov. 1989.

[11] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[12] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 751-772, Oct. 1971.

[13] Berrou, C.; Glavieux, A.; Thitimajshima, O. "Near Shannon limit error-correcting coding and decoding: Turbo-codes", *Proc. ICC'93*, pp.1064-70, 1993.

[14] Massey, P. C.; Takeshita, O. Y.; Costello, D. J. Jr. "Contradicting a Myth: Good Turbo Codes With Large Memory Order", *Proceedings of the International Symposium on Information Theory*, pp. 122, June 2000.

[15] Pérez, L. C.; Seghers, J.; Costello, D. J. Jr. "A Distance Spectrum Interpretation of Turbo Codes", *IEEE Trans. on Inform. Theory*, Vol. 42, No. 6, November 1996.

[16] Benedetto, S.; Montorsi, G. "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Trans. on Inform. Theory*, Vol. 42, No. 2, March 1996.

[17] Takeshita, O. Y.; Collins, O. M.; Massey, P. C.; Costello, D. Jr. "A Note on Asymmetric Turbo-Codes", *IEEE Communications Letters*, Vol. 3, No. 3, November 1999.

[18] Dolinar, S.; Divsalar, D. "Weight Distribution for Turbo Codes Using Random and Nonrandom Permutations", *TDA Progress Report*, 42-122, August 15, 1995.

[19] ten Brink, S., "Designing Iterative Decoding Schemes with the Extrinsic Information Transfer Chart", *AEU International Journal of Electronics and Communications*, Vol. 54, No. 6, pp. 389-98, November 2000.

[20] Palazzo, R. Jr. "Analysis of Periodic Linear and Nonlinear Trellis Codes", Ph.D Dissertation, University of California, Los Angeles.

[21] C. E. Shannon, "The mathematical theory of communication," *BSTJ*, vol. 28, pp. 379-423, Oct. 1949.

[22] T. M. Cover, and J. A. Thomas, *Elements of Information Theory*. New York: John Wiley, 1991.

[23] S. Vembu and S. Verdu, "The source-channel separation theorem revisited," *IEEE Trans. Inform. Theory,* vol. IT-41, pp. 44-54, Jan. 1995.

[24] T. C. Ancheta, Jr., "Joint source channel coding," Ph.D. dissertation, Univ. Notre Dame, Notre Dame, IL, Aug. 1977.

[25] J. L. Massey, "Joint source and channel coding," in *Communication Systems and Random Process Theory,* J. K. Skwirzynski, Ed. The Netherlands: Sijthoff and Nordhoff, 1978, pp. 279-293.

[26] J. G. Dunham and R. M. Gray, "Joint source and channel trellis encoding," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 516-519, July 1981.

[27] E. Ayanoglu and R. M. Gray, "The design of joint source and channel trellis waveform coders," *IEEE Trans. Inform. Theory,* vol. IT-33, pp. 855-865, July 1987.

[28] J. W. Modestino, D. G. Daut, and A. L. Vickers, "Combined source channel coding of images using the block cosine transform," *IEEE Trans. Commun.,* vol. COM-29, pp. 1262-1274, Sept. 1981.

[29] J. W. Modestino, V. Bhaskaran, and J. B. Anderson, "Tree encoding of images in the presence of channel errors," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 667-697, Nov. 1981.

[30] D. Comstock and J. D. Gibson, "Hamming coding of DCT compressed images over noisy channels," *IEEE Trans. Commun.,* vol. COM-32, pp. 856-861, July 1984.

[31] C. C. Moore and J. D. Gibson, "Backward adaptive lattice and transversal predictors in ADPCM," *IEEE Trans. Commun.,* vol. COM-33, pp. 74-82, Jan. 1985.

[32] D. J. Goodman and C. E. Sundberg, "Combined source and channel coding for variable bit-rate speech transmission," *Bell Syst. Tech. J.,* vol. 62, pp. 2735-2764, Nov. 1983.

[33] C.-E. Sundberg, "The effect of single bit errors in standard PCM systems," *IEEE Trans. Commun.,* vol. COM-24, pp. 1062-1064, June 1976.

[34] M. Srinivasan, R. Chellapa, and P. Burlina, "Adaptive source-channel subband video coding for wireless channels," *Proc. IEEE ICIP,* pp. 85-88, Oct. 1995.

[35] K.-P. Ho and J. M. Kahn, "Transmission of analog signals using multicarrier modulation: a combined source-channel coding approach," *IEEE Trans. Commun.,* vol. COM-44, pp. 1432-1443, Nov. 1996.

[36] A. J. Kurtenbach and P. A. Wintz, "Quantizing for noisy channels," *IEEE Trans. Commun. Technol.,* vol. COM-17, pp.291-302, Apr. 1969.

[37] N. Farvardin and V. Vaishampayan, "Optimal quantizer design for noisy channels: an approach to combined source-channel coding," *IEEE Trans. Inform. Theory,* vol. IT-33, pp. 827-838, Nov. 1987.

[38] K.-Y. Chang and R. W. Donaldson, "Analysis, optimization, and sensitivity study of differential PCM systems operating on noisy communication channels," *IEEE Trans. Commun.,* vol. COM-20, pp. 338-350, June 1972.

[39] R. Steele, D. J. Goodman, "Detection and selective smoothing of transmission errors in linear PCM," *Bell Syst. Tech. J.*, vol. 56, pp. 399-409, Mar. 1977.

[40] R. Steele, D. J. Goodman, and C. A. McGonegal, "A difference detection and correction scheme for combating DPCM transmission errors," *IEEE Trans. Commun.*, vol. COM-27, pp. 252-255, Jan. 1979.

[41] K. N. Ngan and R. Steele, "Enhancement of PCM and DPCM images corrupted by transmission errors," *IEEE Trans. Commun.*, vol. COM-30, pp. 257-269, Jan. 1982.

[42] G. H. Pitt, III, L. Swanson, and J. H. Yuen, "Image statistics decoding for convolutional codes," Tech. Rep. TDA Progress Rep. 42-90, JPL, Pasadena, CA, Apr.-June 1987.

[43] R. C. Reininger and J. D. Gibson, "Soft decision demodulation and transform coding," *IEEE Trans. Commun.*, vol. COM-31, pp. 572-577, Apr. 1983.

[44] K. Sayood and J. C. Borkenhagen, "Utilization of correlation in low rate DPCM systems for channel error protection," in *Proc. IEEE ICC,* June 1986, pp. 1888-1892.

[45] K. Sayood and J. C. Borkenhagen, "Use of residual redundancy in the design of joint source channel coders," *IEEE Trans. Commun.*, vol. 39, pp. 838-846, Mar. 1991.

[46] K. Sayood, F. Liu, and J. D. Gibson "A constrained joint source/channel coder design," *IEEE J. Select. Areas in Commun.*, vol. 12, pp. 1584-1593, Dec. 1994.

[47] M. E. Hellman, "On using natural redundancy in the design of joint source channel coders," *IEEE Trans. Commun.*, vol. COM-22, pp. 1690-1693, Oct. 1974.

[48] M. E. Hellman, "Convolutional source encoding," *IEEE Trans. Inform. Theory,* vol. IT-21, pp. 651-656, Nov. 1975.

[49] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84-95, Jan. 1980.

[50] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Norwell, MA: Kluwer Academic Publishers, 1991.

[51] J. R. B. De Marca and N. S. Jayant, "An algorithm for assigning binary indices to the code vectors of a multidimensional quantizer," in *Proc. IEEE Int. Comm. Conf.*, Seattle, WA, pp. 1128-1132, June 1987.

[52] K. A. Zeger and A. Gersho, "Zero redundancy channel coding in vector quantization," *IEEE Electron. Lett.*, vol. 23, pp. 654-655, June 1987.

[53] H. Kumazawa, M. Kasahara, and T. Namekawa, "A construction of vector quantizers for noisy channels," *Electronics and Engineering in Japan,* vol. 67-B, no. 4, pp. 39-47, 1984.

[54] N. Farvardin, "A study of vector quantization for noisy channels," *IEEE Trans. Inform. Theory,* vol. IT-36, pp. 799-809, July 1990.

[55] N. Farvardin and V. Vaishampayan, "On the performance and complexity of channel-optimized vector quantizers," *IEEE Trans. Inform. Theory,* vol. IT-37, pp. 155-160, Jan. 1991.

[56] N. Phamdo, N. Farvardin and T. Moriya, "A unified approach to tree-structured and multistage vector quantization for noisy channels," *IEEE Trans. Inform. Theory,* vol. IT-39, pp. 835-850, May 1993.

[57] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits and Systems for Video Tech.,* vol. 6, June 1996.

[58] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing,* vol. 1, pp. 205-220, April 1992.

[59] P. G. Sherwood and K. Zeger, "Progressive image coding for noisy channels," *IEEE Signal Processing Letters,* vol. 4, No. 7, July 1997.

[60] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.,* vol. COM-36, pp. 389-400, Apr. 1988.