

# Iterative Viterbi Algorithm: Algorithm and Implementation

-A simple Turbo decoding method using VA decoding hardware

Lei Wei, *Senior Member, IEEE*

**Abstract-** In this paper, we present a simple Turbo decoding method called the iterative Viterbi algorithm (IVA). The IVA uses the Viterbi algorithm (VA) hardware to achieve near Turbo code performances. We discuss the IVA for two typical cases: single parity concatenation and double parity concatenation. We show that 3-bit branch metric quantization, 7 or 8 bit state metric precision and a survivor length of five times the constraint length yields little degradation for the IVA. Our results show that without changing the VA hardware (except adding some additional circuits) the error performance of several standard systems can be significantly improved.

**Index Terms--**Turbo Decoding, Coding for Wireless Systems, Iterative Decoding, Viterbi decoding, Serial Concatenated Codes

## I. INTRODUCTION

Turbo codes have completely revolutionized the field of error control coding [1] [2]. Recently, many iterative decoding algorithms have been proposed or rediscovered for decoding compound codes that are composed of a collection of interacting constituent codes. Examples of such compound codes and decoding algorithms include: serially concatenated codes [3], [4], parallel concatenated codes [1], [2], [5], Gallager's low-density parity-check codes [6], [7], [8], [9], various product codes [10], parity-concatenated codes

---

Dr Lei WEI is with School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, NSW 2522, Australia. This work is partly supported by Large ARC Grant R00107858.

[11]-[16], and multilevel codes[17], [18]. In all these cases, near Shannon-limit performance has been achieved.

Over the last 30 years, the Viterbi algorithm (VA) [19], [20] has been widely applied in digital communications. Manufacturers have accumulated many years of experience in not only building the VA circuits for variant situations (see [21]-[27] as examples), but also building other circuitry such as synchronization circuits and carrier tracking and estimation circuits to support the VA device. Can we use these VA circuits to perform iterative decoding? Our recent works provide a positive answer for this question [11]-[16]. The new algorithm is called iterative Viterbi algorithm (IVA). In this paper, we will summarize our recent works and discuss implementation issues regarding the IVA decoder.

The IVA was originated from the bootstrap iterative decoding method developed by Jelinek and Cocke [28], [29]. In [30], Cabral, Costello and Chevillat noted significant similarity between the turbo decoding method and the bootstrap iterative decoding method. In [31] Wei extended the results of [28]-[30] to near optimally decode parity-concatenated convolutional codes. One of the simplified bootstrap algorithms, which only uses the Viterbi algorithm, was reported in [12] and [16], and is now called the iterative Viterbi algorithm (IVA). In [13]-[15], the IVA is extended to decode parity-concatenated trellis codes and excellent performance has been achieved.

The paper is organized as follows. In Section II we review several typical concatenated error control coding structures that are commonly used in practice. In Section III we show how to modify these structures to obtain parity concatenated codes. In Section IV, we present the IVA algorithms for these codes. Section V discusses the implementation issues. In Section VI we give several numerical results and lastly, Section VII concludes. For the sake of clarity, we focus on Additive White Gaussian Noise (AWGN) channels only.

## II. SEVERAL POPULAR SERIAL CONCATENATED CODES

### A. Concatenation Using CRC and Convolutional Codes

Concatenated codes, using a cyclic redundancy check (CRC) code as the outer code and a convolutional code as the inner code, are very popular in the wireless communications (see GSM and IS-95 CDMA systems as examples). Figure 1 shows the block diagram of such a concatenated code, where the CRC code performs error detection and the convolutional code is used for error correction.

Assume that the output of an information source is a sequence of the binary digits “0” and “1”. At the CRC encoder, this binary information sequence is segmented into blocks of a few hundreds of bits each (eg. 172 information bits in the IS-95 system). Let  $k_b$  denote the length of information bits per block. Each block is fed into the CRC encoder (eg. one of the generator polynomials used in IS-95 is  $g(D) = D^{12} + D^{11} + D^{10} + D^9 + D^8 + D^4 + D + 1$ ). The encoder generates a codeword of  $n_b$  bits, which includes  $n_b - k_b$  redundant bits used for error detection. At the end of this block we often insert  $m_c$  zero bits as the tail bits, where  $m_c$  is the number of shift registers in the convolutional encoder. We then feed the block of  $n_b + m_c$  bits into a rate  $k_c/n_c$  convolutional encoder, where  $k_c$  is the number of information bits per unit time that enter the inner encoder and  $n_c$  is the number of transmission bits per unit time that output from the encoder. The convolutional encoder generates  $(n_b + m_c)n_c/k_c$  bit block for transmission over radio channels. Using the IS-95 system as an example, we have  $n_b = 184$ ,  $m_c = 8$ ,  $k_c = 1$ ,  $n_c = 2$  and  $(n_b + m_c)n_c/k_c = 384$ .

### B. Concatenation Using Block and Convolutional Codes

Concatenated codes use a block code (such as a Reed-Solomon (RS) code or a BCH code) as the outer code and a trellis code (such as a convolutional code or a TCM signal) as the inner code. These codes are very

---

<sup>1</sup> The encoder structures for CRC and convolutional codes can be found in [34] in details.

popular in data communications (in particular deep-space communication systems, cable modem systems and digital TV systems as examples). Figure 2 shows the block diagram of such a concatenated code, where a convolutional code is used to perform initial error correction and a BCH code is used for further correcting the residual erroneous bits.

Again, we assume that the output of an information source is a sequence of the binary digits “0” and “1”. At the BCH encoder, the binary information sequence is segmented into blocks of  $k_b$  bits. Each block is fed into a  $(n_b, k_b)$  BCH encoder, which generates a codeword of  $n_b$  bits.  $I$  consecutive codewords are then stored by row in an  $I \times n_b$ -bits interleaver, and are read out by column bit-by-bit. These bits are then fed into a rate  $k_c/n_c$  convolutional encoder  $k_c$  bits at a time. At each time, the encoder produces  $n_c$  bits. For each  $I \times k_b$  information bit block, the concatenated encoder produces a  $I \times n_b \times n_c / k_c$  bit block for transmission.

### III. CONVERTING THE ABOVE STRUCTURES TO PARITY CONCATENATED CODES

Without making any modifications to the code structures as presented in Section II, it is possible to obtain some limited improvements of the error performance if we replace the VA decoder by the IVA decoder [12]. In order to obtain a marked improvement, it is necessary to replace powerful block codes by some simple parity-check codes and use tail-biting trellis [32]. The analytical justification for this modification can be found in [12]. In this section, we will show how to modify these structures.

#### A. Converting the CRC Concatenated Code to a Single Parity-check Concatenated Code.

Two changes are needed to convert the CRC concatenated code to a single parity-check concatenated code namely,

- (a) *Terminate the convolutional encoder using the tail-biting method [32]. For the feed-forward encoder, we feed the last  $m_c$  bits of the block into the shift registers before starting to encode;*
- (b) *Replace the CRC code by a simple parity code, that is,  $g(D) = D^{n_b - k_b} + 1$ .*

For the above IS-95 system, we can select  $n_b = 192$  and  $k_b = 176$ . Figure 3 (b) gives a simple example where  $k_b = 4$ ,  $n_b = 6$ . That is, the information bit sequence, 1011, is firstly encoded using the parity-check code  $g(D) = D^2 + 1$  to a sequence, 101101. After that, the encoded sequence is then fed into the convolutional encoder one bit at a time. The convolutional encoder produces the codeword, 011011011011, for transmission.

### B. Converting the BCH Concatenated Code to a Double Parity-check Concatenated Code.

Now we show how to construct a double parity-check concatenated code. A simple example can be found in Figure 3 (c), where  $I = 3$ ,  $k_b = 4$ ,  $n_b = 6$ .

Here, three modifications are needed.

(a) Store  $I - 1$  consecutive block codewords of  $k_b$  bits each by row in the first  $I - 1$  rows in an  $I \times k_b$ -bits interleaving buffer. The  $I$ -th row is then constructed as follows. Each bit of the  $I$ -th row is the even parity-check bit of the bits of the first  $I - 1$  rows in the corresponding column (see the parity row just before the encoder  $g(D) = D^2 + 1$  in Figure 3 (c)).

(b) Replace the BCH code by a simple parity code, that is,  $g(D) = D^{n_b - k_b} + 1$ . In Figure 3 (c) it is  $D^2 + 1$ . Feed the bit-blocks row-by-row into the encoder one bit at a time and generate  $I$  output codewords of  $n_b$  bits each.

(c) These  $I$  codewords are then fed into the convolutional encoder row-by-row one bit at a time. Each row forms a tail-biting trellis. That is, we feed the last  $m_c$  bits of the codeword row into the shift registers of the encoder before starting to encode every row.

## IV. IVA FOR PARITY CONCATENATED CODES

In this section, we present the IVA for single and double parity-check concatenated codes.

Assume the output bit block ( $\mathbf{z}$ ) from the convolutional encoder is modulated into BPSK signals ( $\mathbf{s}$ ) and then transmitted over a AWGN channel, that is, the received signal  $r_i = s_i + n_i$ , where  $s_i = \sqrt{E_b R}(2z_i - 1)$ ,  $n_i$  is a zero mean independent Gaussian Variable with variance  $N_0/2$  (double sided),  $E_b$  is the energy per information bit and  $R$  is the overall rate of the code. For the single parity concatenated code  $R = k_b k_c / (n_b n_c)$  and for the double parity concatenated code we have  $R = k_b k_c (I - 1) / (n_b n_c I)$ .

The branch metric for the VA is

$$\omega_i(z_i) = -\log P(r_i | z_i) \quad (1)$$

The key concept of the IVA is to utilize the hard decision of the VA decoder and the parity-check relationship between bits introduced by the parity code to modify the branch metric using the VA decoder. Then, feed those modified branch metrics into the VA decoder for next iteration decoding. The flow diagram can be found in Figure 4. The following sections are dedicated to describe the IVA for both single and double parity-concatenated codes.

#### A. IVA for Single Parity Concatenated Codes

The iterative Viterbi algorithm for single parity check concatenated codes is discussed in this section. A simple example is given in Figure 5 to illustrate the procedure. This example is identical to the example given in Figure 3 (b). The branch metrics for  $z_i = 0$   $i=0, 1, \dots, 11$  are [2, 5, 5, 0, 3, 4, 4, 3, 6, 1, 6, 7], respectively; for  $z_i = 1$   $i=0, 1, \dots, 11$  are [5, 2, 2, 7, 4, 3, 3, 4, 1, 6, 1, 0], respectively. For simplicity the metric values in Figure 5 are the differences between  $\omega_i(z=0)$  and  $\omega_i(z=1)$ , i.e.,  $\omega_i = \omega_i(0) - \omega_i(1)$ . Therefore, we have  $\mathbf{R} \rightarrow \omega$ : {-3,3}{3,-7}{-1,1}{1,-1}{5,-5}{5,7} in Figure 5 (a).

### The Iterative Viterbi Algorithm

**Step 1:** Compute the branch metrics using Eqn. (1).

**Step 2:** Apply the standard VA.

At the first iteration, the standard VA is applied to decode the tail-biting convolutional codeword, using the branch metrics given in Eqn. (1). For the example case (in Figure 5 (b)), we can see that the third bit is not decoded correctly in the first iteration.

**Step 3:** Check whether the VA decoding procedure can be terminated. If yes, then terminate the decoding process. If no, then go to step 4.

After VA decoding of each iteration the decoding procedure may be terminated if the decoded codeword is a valid block codeword (given the number of parity bits is reasonable large) or if the number of iterations reaches to a pre-set maximum value. Part (c) in Figure 5 shows that the decoding result is erroneous, thus go to the next iteration.

**Step 4:** Update the branch metric and then go to step 2.

After the first iteration we first modify the branch metric for every bit and then decode the tail-biting convolutional encoded codeword using the VA decoder using the modified branch metric (say  $\omega_i^*$ ).

Clearly, the key steps of the IVA are how to update the branch metric. Now let us see how to update the branch metric of bit  $z_i$ .

Lining-up  $n_b n_c / k_c$  coded bits, we know that [12]

$$z_{i^*} \oplus z_{i^*+a} \oplus \dots \oplus z_{i^*+ab} = 0 \quad (2)$$

where  $i^* = i \bmod a$ ,  $a = n_c(n_b - k_b) / k_c$ ,

$b = \left\lfloor \frac{n_b n_c / k_c - i^* - 1}{a} \right\rfloor$ , and  $\lfloor x \rfloor$  denotes the largest integer no greater than  $x$ . Now let  $h_j = z_{i^*+ja}$  and

$\hat{h}_j = \hat{z}_{i^*+ja}$  where  $j=0, \dots, b$ , and  $\hat{z}_i$  denotes the hard-decision of bit  $z_i$  at the previous iteration.

**Stage 1:** Randomly select an integer (say  $l$ ) between 0 and  $b$ , and  $la + i^* \neq i$ .

**Stage 2:** Compute

$$\hat{W} = \hat{h}_0 \oplus \hat{h}_1 \oplus \dots \oplus \hat{h}_b \oplus \hat{z}_i \oplus \hat{h}_l \quad (3)$$

**Stage 3:** Then the updated branch metric for bit  $z_i$  is

$$\omega_i^*(q) = \omega_i(q) + \lambda \omega_{la+i}^*(q \oplus \hat{W}) \quad (4)$$

where  $\lambda$  is a positive value which is used to control the error propagation and  $q = 0$  or  $1$ . Typically, we have  $\lambda=0.25$ . If the branch metric is quantized, then we can select a non-linear mapping table (eg Tables 1 or 2). The term of  $\lambda \omega_{la+i}^*(q \oplus \hat{W})$  is called the extrinsic branch metric. Update the branch metrics for all bits.

Now, as an example, let us see how to obtain the results in the 4<sup>th</sup> column in Step 4 in Figure 5 (c). For this case, we have  $a = 4$  and  $b = 2$ . Line-up 12 coded bits as  $[z_0 z_4 z_8]$ ,  $[z_1 z_5 z_9]$ ,  $[z_2 z_6 z_{10}]$  and  $[z_3 z_7 z_{11}]$  for  $i^* = 0, 1, 2, 3$  respectively. After the first iteration, we have  $[\hat{z}_0 \hat{z}_4 \hat{z}_8] = [0 0 1]$ ,  $[\hat{z}_1 \hat{z}_5 \hat{z}_9] = [1 0 0]$ ,  $[\hat{z}_2 \hat{z}_6 \hat{z}_{10}] = [1 1 1]$  and  $[\hat{z}_3 \hat{z}_7 \hat{z}_{11}] = [0 1 1]$ . Now let us update the metric for bits  $z_4$  and  $z_5$ . For  $z_4$  we have  $i=4$ , thus  $i^* = 0$ .

Stage 1: we select  $l = 2$ .

Stage 2:  $\hat{W} = \hat{h}_0 \oplus \hat{h}_1 \oplus \hat{h}_2 \oplus \hat{z}_i \oplus \hat{h}_2 = \hat{z}_0 = 0$ .

Stage 3:  $\omega_{la+i}^*(0) = \omega_8(0) = 6$  and  $\omega_{la+i}^*(1) = \omega_8(1) = 1$

According to Table we have  $\lambda \omega_{la+i}^*(0) = \lambda \omega_8(0) = 1$  and  $\lambda \omega_{la+i}^*(1) = \lambda \omega_8(1) = 0$ . Since

$\omega_i^*(q) = \omega_i(q) + \lambda \omega_{la+i}^*(q)$  for  $q=0, 1$ , we obtain  $\omega_4^* = \omega_4^*(0) - \omega_4^*(1) = 0$ .

For  $z_5$  we have  $i=5$ , thus  $i^* = 1$ .

Stage 1: we select  $l = 2$ .

Stage 2:  $\hat{W} = \hat{h}_0 \oplus \hat{h}_1 \oplus \hat{h}_2 \oplus \hat{z}_i \oplus \hat{h}_2 = \hat{z}_1 = 1$ .

Stage 3:  $\omega_{la+i}^*(0) = \omega_9(0) = 1$  and  $\omega_{la+i}^*(1) = \omega_9(1) = 6$

Using Table we have  $\lambda\omega_{la+i}^*(0) = \lambda\omega_9(0) = 0$  and  $\lambda\omega_{la+i}^*(1) = \lambda\omega_9(1) = 1$ . Since  $\omega_i^*(q) = \omega_i(q) + \lambda\omega_{la+i}^*(q \oplus 1)$  for  $q=0, 1$ , we have  $\omega_5^* = \omega_5^*(0) - \omega_5^*(1) = \omega_5(0) + \lambda\omega_9(1) - \omega_5(1) - \lambda\omega_9(0) = 2$ .

The same procedure is applied to all other metrics. We then obtain the new and updated metrics listed in Figure 5 (d). After the VA, the decoded sequence satisfies the parity check condition, thus the decoding procedure is terminated.

In this example, we can see that after modifying the branch metric the correct codeword becomes the ML candidate. Thus, the erroneous bit in the first iteration is corrected in the second iteration.

**Table (a)**

$\omega$	0	1	2	3	4	5	6	7
$\lambda\omega$	0	0	0	0	0	1	1	1

**Table (b)**

$\omega$	0	1	2	3	4	5	6	7
$\lambda\omega$	0	0	0	0	1	1	1	2

### B. IVA for Double Parity Concatenated Codes

The IVA for double parity concatenated codes is largely identical to the IVA for single parity concatenated codes, except each updated branch metric is the sum of three terms: its own branch metric, the *extrinsic* branch metric from the parity-check relationship in row, and the *extrinsic* branch metric coming from the parity-check relationship in column. In the first iteration, the standard VA is applied to decode

each tail-biting row consecutively until the last row (i.e., the  $I$ -th row) has been decoded. During this iteration the branch metrics given in (1) are used and the parity-check constraints in row and in column are ignored.

At the end of each iteration we may terminate the decoding procedure if the decoded codeword is a valid block codeword (that is, satisfies the parity-check constraints in row and in column) or if the number of iterations reaches a maximum threshold value.

If the decoding procedure needs to continue, then we first update the branch metric for each bit based on the parity-check relationships between the bits in row and in column. Next, we apply the standard VA again to decode each tail-biting row. The updated branch metrics are now used.

The branch metric updating procedure is given as follow. We use bit  $z_{i,j}$  (i.e., the bit at the  $i$ -th column and the  $j$ -th row) as an example.

The updated branch metric for bit  $z_{i,j}$  is

$$\omega_{i,j}^*(q) = \omega_{i,j}(q) + \lambda\omega_{la+i^*,j}(q \oplus \hat{W}) + \lambda\omega_{i,j^\#}(q \oplus \hat{W}^\#) \quad (5)$$

The way to compute  $\lambda\omega_{la+i^*,j}(q \oplus \hat{W})$  is identical to  $\lambda\omega_{la+i^*}(q \oplus \hat{W})$  in (4). So, we will focus only on how to compute  $\lambda\omega_{i,j^\#}(q \oplus \hat{W}^\#)$ .

Lining-up  $I$  bits in the column corresponding to bit  $z_j$ , due to the column parity-check code we know that  $z_{i,0} \oplus z_{i,1} \oplus \dots \oplus z_{i,I-1} = 0$ . Randomly select an integer (say  $j^\#$ ) between 0 and  $I-1$ , and  $j^\# \neq j$ . Computing  $\hat{W}^\# = \hat{z}_{i,0} \oplus \hat{z}_{i,1} \oplus \dots \oplus \hat{z}_{i,I-1} \oplus \hat{z}_{i,j} \oplus z_{l,j^\#}$ , we then obtain  $\lambda\omega_{i,j^\#}(q \oplus \hat{W}^\#)$ .

## V. IMPLEMENTATION ISSUES

Several implementation issues will be addressed in this section. These include namely branch metric quantization and state metric overflow problem.

### A. Precision for branch metrics

As shown in [21], each branch metric can be quantized into a 3-bit value (0 to 7). It is well known (see [21]) that 3-bit quantization and a survivor length of five times the constraint length yields little degradation (about 0.2-0.3dB) from the theoretical limit, when the VA is applied. In our study, we found the IVA also only needs 3-bit quantization and a survivor length of five times the constraint length for a degradation of 0.3 dB from the theoretical limit. Thus, nothing needs to change in the branch metric quantization and the survivor length in the VA decoding circuits.

If we select Table (a) then the extrinsic metric is either 0 or 1. If we select Table (b), then the extrinsic metric could be 0, or 1, or 2.

### B. State metric overflow problem

The recursive state metric update in the VA results in unbounded growth of state metrics due to the addition of branch metrics, which are always non-negative. Two common methods are often used to deal with state metric overflow problem [33]. Both methods are optimal, that is, they will not cause any performance degradation.

The first method is the traditional state metric rescaling. That is, after each decoding step, all state metrics subtract the smallest state metric. As shown in [33] the difference between state metrics is no larger than  $m_c n_c B$ , where  $B$  is the largest value of the branch metric per transmission bit. If we select 3-b branch metric quantization, then  $B=7$ . Thus, for any rate  $\frac{1}{2}$  code with  $m_c = 2, 3, 4, 5, 6, 7, 8$ ,  $m_c n_c B = 28, 42, 56, 70, 84, 98, 112$  respectively. Therefore, the required state metric precision is no more than 5-b, 6-b, 6-b, 7-b, 7-b, 7-b, 7-b, for  $m_c = 2, \dots, 8$ , respectively, where the symbol, “-b”, denotes “bits”.

For a single parity concatenated code using Table (a), we have  $B = 8$ , thus the required state metric precision is **6-b**, 6-b, 6-b, 7-b, 7-b, 7-b, **8-b** for any rate  $\frac{1}{2}$  convolutional code with  $m_c = 2, \dots, 8$ ,

respectively. For a double parity concatenated code using Table (a), we obtain  $B = 9$ , thus the required state metric precision could reach **6-b**, 6-b, **7-b**, 7-b, 7-b, 7-b, **8-b** for any rate  $\frac{1}{2}$  convolutional code with  $m_c = 2, \dots, 8$ , respectively. That is, for  $m_c = 2, 4, 8$  we might need to increase the precision of state metrics by 1 bit for the IVA. However, for a particular code state metric precisions might not need to increase. For example, we can easily show that the IVA does not require to increase the state metric precisions for the following three rate  $\frac{1}{2}$  convolutional codes: (a)  $g^{(1)} = 5$ ,  $g^{(2)} = 7$  and  $m_c = 2$ ; (b)  $g^{(1)} = 46$ ,  $g^{(2)} = 72$  and  $m_c = 4$ ; (c)  $g^{(1)} = 753$ ,  $g^{(2)} = 561$  and  $m_c = 8$ .

For the double parity concatenated code using Table (b), we obtain  $B = 10$ . For the rate  $\frac{1}{2}$  convolutional code with  $g^{(1)} = 744$ ,  $g^{(2)} = 554$  and  $m_c = 6$ , the required state metric precision is 7-b. Thus, we do not need to change the precisions of branch metric and state metric of the VA decoder.

The second method is the two's complement arithmetic approach presented in [33]. In the second method, the number of bits required for the state metric is given as

$$c \geq \log_2[(m_c + 1)n_c B + 1] + 1 \quad (6)$$

If we select 3-b branch metric quantization, then  $B=7$ . Thus, for any rate  $\frac{1}{2}$  convolutional code with  $m_c = 2, 3, 4, 5, 6, 7, 8$ , the required state metric precision is 7-b, 7-b, 8-b, 8-b, 8-b, 8-b, 8-b respectively.

For a single parity concatenated code using Table (a), we obtain  $B = 8$ , thus the required state metric precision is 7-b, **8-b**, 8-b, 8-b, 8-b, **9-b**, **9-b** for any rate  $\frac{1}{2}$  convolutional code with  $m_c = 2, \dots, 8$ , respectively. However, for the rate  $\frac{1}{2}$  codes: (a)  $g^{(1)} = 74$ ,  $g^{(2)} = 64$  and  $m_c = 3$ ; (b)  $g^{(1)} = 712$ ,  $g^{(2)} = 476$  and  $m_c = 7$ ; (c)  $g^{(1)} = 753$ ,  $g^{(2)} = 561$  and  $m_c = 8$ , we find the required state metric precision is still 7-b, 8-b and 8-b, respectively.

For a double parity concatenated code with using Table (a), we obtain  $B=9$ , thus the required state metric precision is 7-b, **8-b**, 8-b, 8-b, 8-b, **9-b**, **9-b** for any rate  $\frac{1}{2}$  convolutional code with  $m_c = 2, \dots, 8$ , respectively. However, for the rate  $\frac{1}{2}$  codes: (a)  $g^{(1)} = 74$ ,  $g^{(2)} = 64$  and  $m_c = 3$ ; (b)  $g^{(1)} = 712$ ,  $g^{(2)} = 476$  and  $m_c = 7$ ; (c)  $g^{(1)} = 753$ ,  $g^{(2)} = 561$  and  $m_c = 8$ , we find the required state metric precision is 7-b, 8-b and **9-b**, respectively. That is, only for code (c) we need to increase 1 bit in the state metric precision.

In summary, when we apply the IVA, the precision of metrics might need to increase for some cases. But for many real applications the precision needs not to change at all. Here let us consider two typical applications, in which we do not need to change the VA decoder in term of the branch metric precision and the state metric precision. The first one is to replace the concatenated code using the convolutional code as the inner code and the Reed-Solomon code as the outer code by the double parity concatenated code using a rate  $\frac{1}{2}$  and  $m_c = 6$  convolutional code. The second one is to replace the VA decoder in the Qualcomm CDMA system by the single parity concatenated code using a rate  $\frac{1}{2}$  and  $m_c = 8$  convolutional code.

## VI. NUMERICAL RESULTS

In this section we will present numerical results for two typical systems. System (a) originates from that used in the forward link in the Qualcomm IS95 system. In the Qualcomm system, we have  $n_b = 192$ ,  $k_b = 172$ , a CRC code with 12 redundant bits, and 8 tail-bits. System (a) is a single parity-check concatenated code with  $n_b = 192$ ,  $k_b = 176$ , and a simple parity code with 16 redundant bits. The convolutional code remains the same, i.e., the rate  $\frac{1}{2}$  code with  $g^{(1)} = 753$ ,  $g^{(2)} = 561$  and  $m_c = 8$ .

Parameters of System (b) come from the concatenated code using (255,223) RS code as the outer code and the rate  $\frac{1}{2}$  convolutional code with  $g^{(1)} = 744$ ,  $g^{(2)} = 554$  and  $m_c = 6$  as the inner code with  $I=2$ . For such a code, a codeword of 4080 bits contains 3568 information bits. System (b) is a double parity-

check concatenated code with  $n_b = 255$  ,  $k_b = 238$  , and  $I = 16$  . System (b) generates a codeword of 4080 bits, but contains 3570 information bits.

We will study the effect of branch metric quantization and state metric precision. For system (a) we use Table (a) and in system (b) we use Table (b). We will also study the effect of the IVA on the error detection capability of the simple parity code.

#### A. System (a)

For system (a) we will study the block error rate and the probability of undetected error. Each simulation trial was terminated if at least 50 error blocks were obtained, or if the total number of blocks processed reached  $3 \times 10^6$ .

In Figure 6 we present the block error rate for system (a). Clearly, an improvement of about 0.9dB can be obtained using the IVA. In Figure 7, we show the undetected block error probability. It clearly shows that the error detection capability of the block code has been weakened. But at a block error rate of 1%, the undetected block error probability is less than  $3 \times 10^{-5}$ . That is, phone users will experience no more than one error block every 11 minutes on average. If we use an additional 2 bits (172 bits in system (a) vs. 170 bits in the Qualcomm system) to reduce this probability, then phone users will only experience no more than one error block every 44 minutes on average.

Furthermore, the number of average iteration for the IVA is 1.2-1.4 times of the VA at a block error rate of 1%. We conclude that the IVA can obtain 0.5 dB - 1 dB over the VA for the Qualcomm system on AWGN. The required computation complexity is about 20% to 40% more than that in the VA.

#### B. System (b)

For system (b) we will focus on bit error rate. However, unlike the turbo codes, the parity-check concatenated code can reliably indicate erroneous blocks of 238 information bits each. Thus, we will also examine the block error rate. Each simulation trial was terminated if at least 50 error blocks were obtained, or if the total number of bits processed reached  $4 \times 10^8$ .

In Figure 8 we show the bit error rate of system (b) as a function of  $E_b / N_o$ . For comparison, we also plot the bit error rates of the VA using the rate  $\frac{1}{2}$  64 state convolutional code and the performance of the RS and CC concatenated code decoded by the VA and Berlekamp decoders. Figure 9 shows the block error probability where each block contains 238 information bits. Figure 10 shows the average number of iterations.

## VII. CONCLUSIONS AND DISCUSSIONS

Recently our research results have shown that superior performance can be achieved by using a simple iterative decoding algorithm called the iterative Viterbi algorithm (IVA) [11]-[16]. Following the Turbo decoding principles, the IVA uses the Viterbi decoder hardware to achieve near Turbo code performances. In this paper we first presented the IVA for two typical cases: single parity concatenation and double parity concatenation. We then showed that, similar to the VA decoder, in several typical applications we studied, 3-bit branch metric quantization, 7 or 8 bit state metric precision and a survivor length of five times the constraint length yields little degradation for the IVA. Our results have shown that without changing the VA hardware (except adding some additional circuits) the error performance of several standard systems can be significantly improved.

## REFERENCES

- [1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correction Coding and Decoding: Turbo Codes," *Proc. IEEE Int. Conf. Commun. (ICC)*, Geneva, Switzerland, 1993, pp.1064-1070.
- [2] C. Berrou and A. Glavieux "Near Optimum Error Correcting Coding and Decoding: turbo-Codes," *IEEE Trans on Communications*, Vol. 44, pp.1261-1271, Oct. 1996.
- [3] G. D. Forney, Jr., *Concatenated codes*, MIT Press, Cambridge, Mass, 1963.

- [4] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. on Inform. Theory*, Vol. 44, pp.909-926, May 1998.
- [5] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. on Inform. Theory*, Vol.42, pp.409-428, Mar. 1996.
- [6] R. G. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MIT Press, 1963, MA, USA.
- [7] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inform. Theory*, Vol. 27, pp. 533-547, Sept. 1981.
- [8] D. J. C. MacKay, "Good codes based on very sparse matrices," *IEEE Trans. on Inform. Theory*, Vol. 45, pp. 399-431, Mar. 1999.
- [9] G. D. Forney, Jr., "On iterative decoding and the two-way algorithm," *Proc. Intl. Symp. Turbo codes and related topics*, Brest, France, Sept. 1997.
- [10] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Inform. Theory*, Vol. 42, pp. 429-445, Mar. 1996.
- [11] Qi Wang, Lei Wei and Rodney A. Kennedy, "Iterative Viterbi Decoding, Trellis Shaping and Multilevel Structure for High-Rate Concatenated TCM," *IEEE Trans. on Communications*, revised and resubmitted.
- [12] L. Wei, "Near Shannon Limit Iterative Viterbi Algorithm for Forney Concatenated Systems," *IEEE Trans. on Inform. Theory*, (submitted, 1999, revising now).
- [13] Q. Wang and L. Wei, "Graph-Based Iterative Decoding Algorithms for Parity-Concatenated Trellis Codes," *IEEE Trans. on Inform. Theory*, to appear Mar. 2001.
- [14] Q. Wang and L. WEI, "Iterative Viterbi algorithm for concatenated multi-dimensional TCM," *Proceeding of ISIT*, p.250, Jun. 25-30, 2000, Sorrento, Italy.

- [15] Q. Wang and L. WEI, "Iterative decoding for parity-concatenated multi-dimensional TCM," *Proceeding of ISITA*, Honolulu, Hawaii, USA, Nov. 5-8, 2000, accepted.
- [16] L. Wei, "On bootstrap iterative Viterbi algorithm," *Proceedings of IEEE ICC'99*, pp.1187-1192, Vancouver, Canada, June 6-9, 1999.
- [17] G. D. Forney, Jr., M.D. Trott, and S.-Y. Chung "Sphere-bound-achieving coset codes and multilevel coset codes," *IEEE Trans. on Inform. Theory*, pp. 820-850, May 2000.
- [18] U. Wachsmann and J. Huber, "Power and bandwidth efficient digital communication using turbo codes in multilevel codes," *Euro. Trans. Telecomm.* Vol. 6, pp. 557-567, Sept. 1995.
- [19] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Inform. Theory*, Vol. IT-13, pp.260-269, No.2, Apr. 1967.
- [20] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, pp.268-278, Mar. 1973.
- [21] J. A. Heller and I. M. Jacobs, "Viterbi decoding for satellite and space communication," *IEEE Trans. on Communications*, Vol. 19, pp. 835-848, Oct. 1971.
- [22] G. Fettweis and H. Meyr, "Parallel Viterbi Algorithm Implementation: Breaking the ACS-Bottleneck," *IEEE Trans. on Communications*, Vol. 37, pp. 785-789, Aug. 1989.
- [23] K. Wen, T. Wen and J. Wang, "A New Transform Algorithm for Viterbi Decoding," *IEEE Trans. on Commuications.*, Vol. 38, pp. 764-772, Jun. 1990.
- [24] O. M. Collins, "The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Decoder," *IEEE Trans. on Communications*, Vol. 40, pp. 1810-1819, Dec. 1992.
- [25] S. Kubota, S. Kato, and T. Ishitani, "Novel Viterbi Decoder VLSI Implementation and its Performance," *IEEE Trans. on Communnications*, Vol. 41, pp. 1170-1178, Aug. 1993.
- [26] P. Black and T. H-Y Meng, "1-Gb/s, Four-State, Sliding Block Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, Vol.32, No.6, pp. 797-805, Jun. 1997.

- [27] I. Kang and A. N. Wilson Jr., "Low-Power Viterbi Decoder for CDMA Mobile Terminals," *IEEE Journal of Solid-State Circuits*, Vol.33, No.3, pp. 473-481, Mar. 1998.
- [28] F. Jelinek and J. Cocke, "Bootstrap hybrid decoding for symmetrical binary input channels," *Information and Control*, Vol. 18, pp. 261-298, Apr. 1971.
- [29] F. Jelinek, "Bootstrap trellis decoding," *IEEE Trans. on Inform. Theory*, Vol. 21, pp. 318-325, May 1975.
- [30] H. A. Cabral, D. J. Costello, Jr., and P.R. Chevillat, "Bootstrap hybrid decoding using the multiple stack algorithm," *Proceeding of IEEE ISIT'97*, pp. 494, Ulm, Germany, June 29 - July 4.
- [31] L. Wei, "Near Optimal Limited Search Algorithms: Part II Convolutional Codes," unpublished, 1997.
- [32] G. Solomon, H. C. A. van Tilborg, "A connection between block and convolutional coded," *SIMA J. of Appl. Math*, pp. 358-369, Vol. 37, No. 2, Oct. 1979.
- [33] A. P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoders," *IEEE Trans. on Communications*, Vol. 37, pp.1096-1100, Nov. 1989.
- [34] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1983

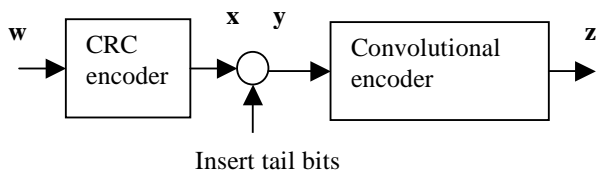


Figure 1 Encoding structure for a concatenated code using CRC and convolutional codes<sup>2</sup>.

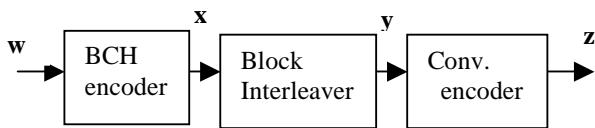
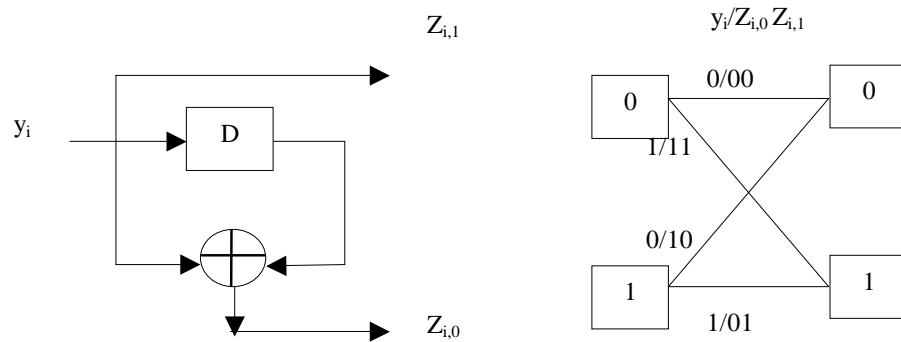
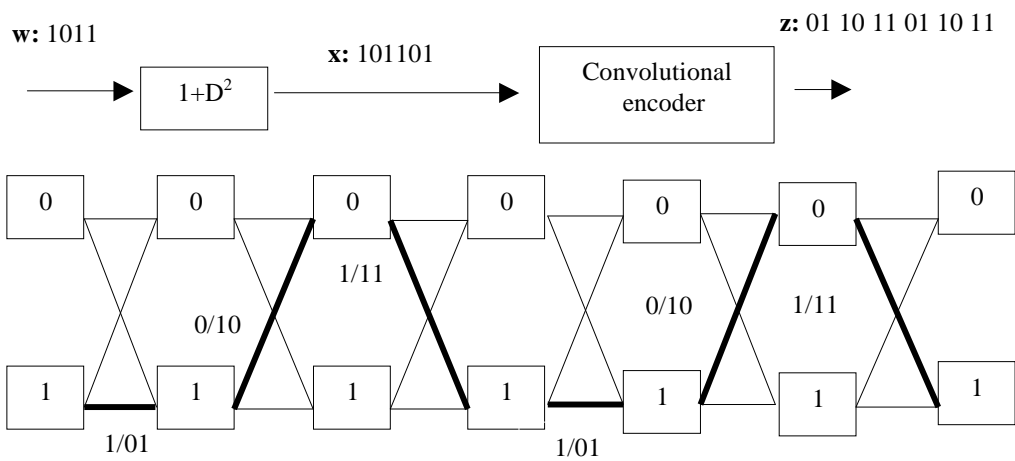


Figure 2 Encoding structure for a concatenated code using a BCH code, a convolutional code and a block interleaver.

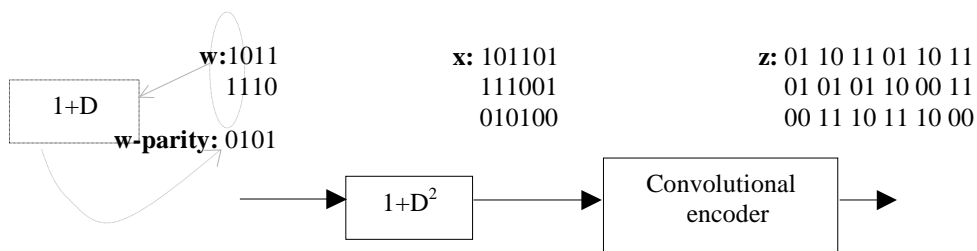
<sup>2</sup> The encoder structures for CRC and convolutional codes can be found in [34] in details.



(a) Encoder and trellis diagram of a rate  $\frac{1}{2}$  convolutional code



(b) An encoder example for a single parity-check concatenated code



(c) An encoder example for a double parity-check concatenated code

Figure 3 Encoding procedures for parity concatenated codes.

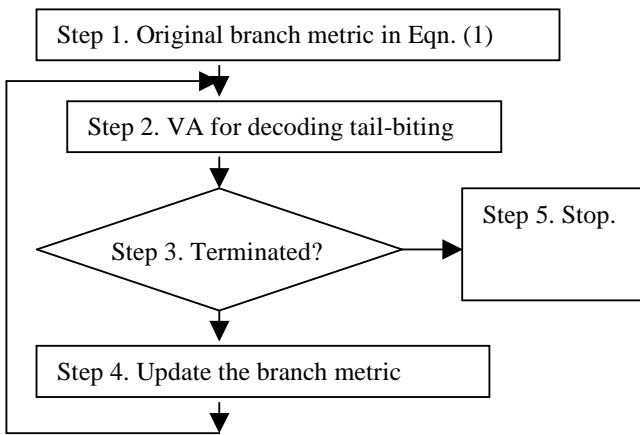
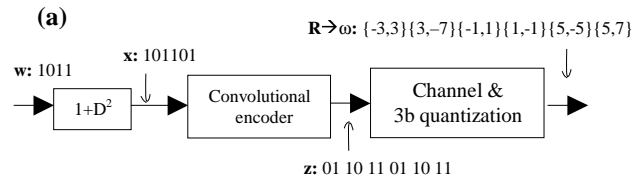


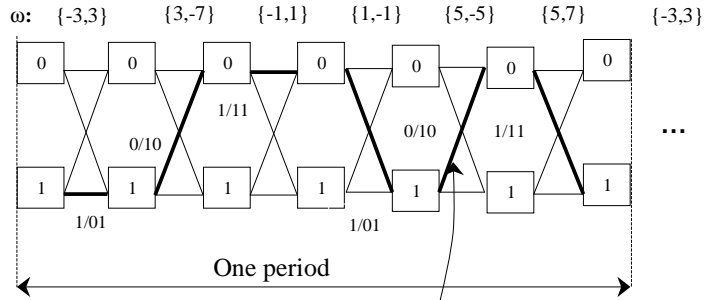
Figure 4 Flow diagram for the IVA



(b) The first iteration:

Step 1: List original branch metric

Step 2: Apply VA for unwrapped trellis



Result after the first iteration is 100101  
(marked as the thickening line)

(c) Step 3: Does the decoding result satisfy the parity check?

No. So go to the second iteration.

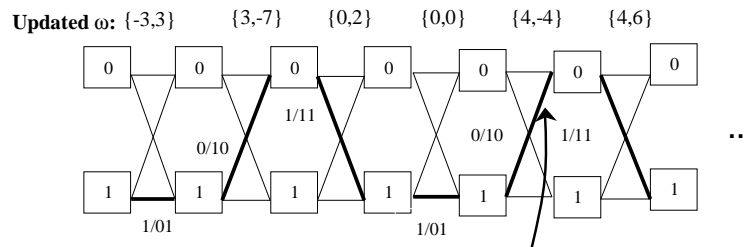
Step 4: Updating all branch metrics.

$l$	1	1	2	2	0	0
$\hat{w}$	10	11	01	10	00	11
$\omega_{l_{a+i}^*}$	-1,1	1,-1	5,-5	5,7	-3,3	3,-7
$\omega_l$	-3,3	3,-7	-1,1	1,-1	5,-5	5,7
$\omega_l^*$	-3,3	3,-7	0,2	0,0	4,-4	4,6

(d) The second iteration:

Step 1: List original branch metric

Step 2: Apply VA for unwrapped trellis



Result after the first iteration is 101101  
(marked as the thickening line)

(e) Step 3: Does the decoding result satisfy the parity check?

Yes. Then go to Step 5 and stop.

Figure 5 An example for decoding the single parity-check code using the IVA.

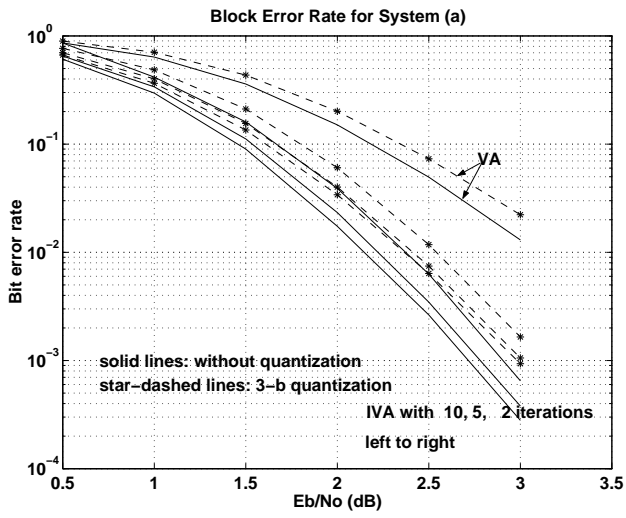


Figure 6 Block error rate for system (a) on AWGN.

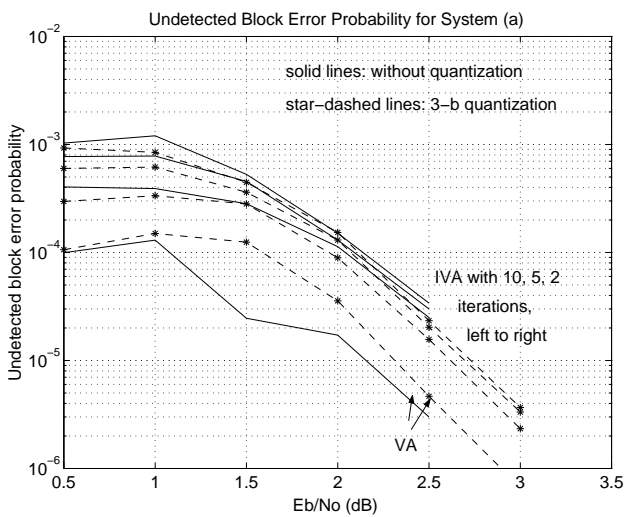


Figure 7 Probability of undetected block error for system (a) on AWGN.

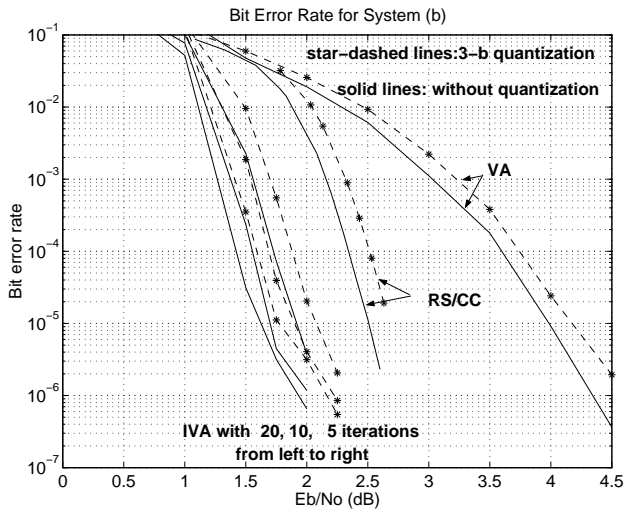


Figure 8 Bit error rate for system (b) on AWGN.

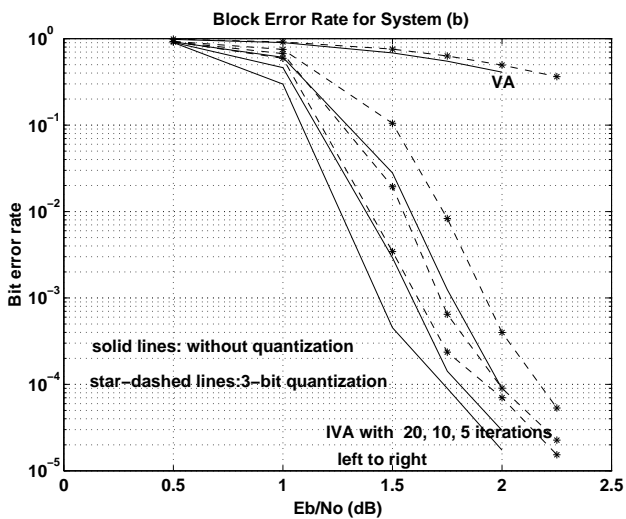


Figure 9 Block error rate for system (b) on AWGN, where each block contains 238 information bits.

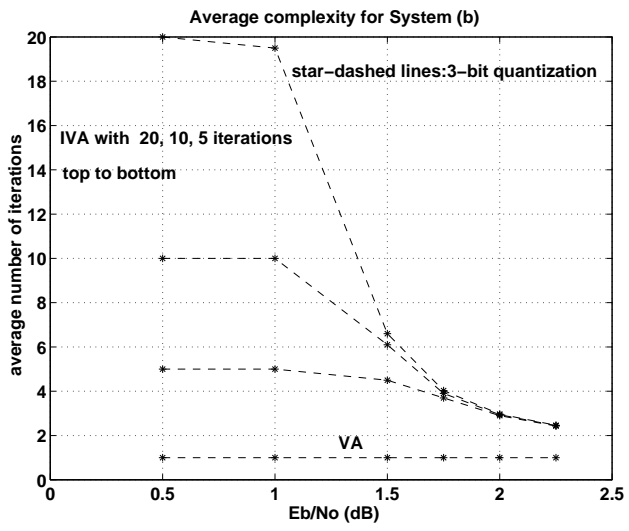


Figure 10 Average number of iterations.