A Simplification of the Modified Bahl Decoding Algorithm for Systematic Convolutional Codes

Steven S. Pietrobon and Adrian S. Barbulescu
Australian Space Centre for Signal Processing
University of South Australia
The Levels SA 5095

Revised 4 January 1996

SUMMARY A soft–in/soft–out algorithm which estimates the a posteriori probabilities (APP) for each transmitted bit is investigated. The soft outputs can be used at the next decoding stage, which could be an outer code or another iteration in an iterative decoding process. This algorithm is estimated to have approximately four times the complexity of the Viterbi algorithm and has the advantage of providing the APP for each decoded bit.

1 INTRODUCTION

The Viterbi algorithm is a maximum likelihood decoding method which minimises the probability of sequence error. However, it does not necessarily minimise the probability of bit error. An algorithm to estimate the *a posteriori probabilities* (APP) of the states and transitions of a Markov source observed through a discrete memoryless channel, was first presented by Bahl et. al. in [1]. This yields the APP for each decoded bit and it is an optimal decoding method for linear codes which minimises the bit error probability. A modified version was presented in [2] where it was used for decoding a new class of convolutional codes, Turbo–Codes, in an iterative process. The derivation presented in [2] led to a very complicated relation to evaluate the APP for each decoded bit.

In this paper we present a simplified APP algorithm which was successfully tested using the rate half systematic convolutional 16 state Turbo code given in [2], with a 20×20 block interleaver of 400 bits. For an E_b/N_0 = 2.0 dB and eight iterations, an output bit error ratio $P_b = 1.5 \times 10^{-4}$ was found.

2 A MODIFIED BAHL ALGORITHM

For an encoder with ν memory cells, we define the encoder state at time k , S_k, as a ν–tuple, depending only on the output of each delay element. The state S_k defined in [2] included d_k, which resulted in an excessively complicated algorithm. The information bit at time k, d_k, is associated with the transition from time k to time k+1 and will change the encoder state from S_k to S_{k+1}. Also suppose that the information bit sequence $\{d_k\}$ is made up of $N-\nu$ independent bits d_k, taking values 0 and 1 with equal probability. We let the encoder initial state S_1 be equal to zero. The last ν information bits ($d_{N-\nu+1}$ to d_N) are set to values that will force the state to 0 at time N+1 (i.e., S_{N+1} = 0). This will slightly reduce the rate of the encoder.

This work was supported in part by the Australian Research Council and under GIRD grant number 17025.

We consider a rate 1/2 systematic feedback encoder whose outputs at time k are the uncoded data bit, d_k, and the coded bit, Y_k. These outputs are modulated with a BPSK or QPSK modulator and sent through an additive white gaussian noise (AWGN) channel. At the receiver end, we define the received sequence:

$$R_1^N = (R_1, \ldots, R_k, \ldots, R_N), \tag{1}$$

where $R_k = (x_k, y_k)$ is the received symbol at time k; x_k and y_k are defined as:

$$x_k = (2d_k - 1) + p_k, \tag{2}$$

$$y_k = (2Y_k - 1) + q_k, \tag{3}$$

with p_k and q_k being two independent normally distributed random variables with variance σ^2. We define the Log Likelihood Ratio, $L(d_k)$ associated with each decoded bit d_k as:

$$L(d_k) = \log\frac{P_r(d_k = 1|R_1^N)}{P_r(d_k = 0|R_1^N)}, \tag{4}$$

where $P_r(d_k = i|R_1^N)$, i = 0,1 is the APP of the data bit d_k. The APP of a decoded data bit d_k can be derived from the joint probability defined by:

$$\lambda_k^i(m) = P_r(d_k = i, S_k = m|R_1^N), \tag{5}$$

and thus the APP of a decoded data bit d_k is equal to:

$$P_r(d_k = i|R_1^N) = \sum_m \lambda_k^i(m), \tag{6}$$

where i = 0,1 and the summation is over all 2^ν states. From (4) and (6), the $L(d_k)$ associated with a decoded bit d_k can be written as:

$$L(d_k) = \log\frac{\sum_m \lambda_k^1(m)}{\sum_m \lambda_k^0(m)}. \tag{7}$$

The decoder can make a decision by comparing $L(d_k)$ to a threshold equal to zero:

$$\hat{d}_k = \begin{cases} 1 \ ; \ L(d_k) \geq 0, \\ 0 \ ; \ L(d_k) < 0. \end{cases} \tag{8}$$

Using Bayes' rule, the joint probability from (5) can be rewritten as follows:

$$\lambda_k^i(m) = \frac{P_r(d_k = i, S_k = m, R_1^k, R_{k+1}^N)}{P_r(R_1^N)}, \tag{9}$$

which can be further expanded to:

$$\lambda_k^i(m) = \frac{P_r(d_k = i, S_k = m, R_1^k) P_r(R_{k+1}^N | d_k = i, S_k = m, R_1^k)}{P_r(R_1^N)}. \tag{10}$$

Taking into account that events after time k are not influenced by that part of the observation up to the time k, (10) can be modified to:

$$\lambda_k^i(m) = \frac{P_r(d_k = i, S_k = m, R_1^k) P_r(R_{k+1}^N | d_k = i, S_k = m)}{P_r(R_1^N)}. \tag{11}$$

We define

$$\alpha_k^i(m) = P_r(d_k = i, S_k = m, R_1^k), \tag{12}$$

$$\beta_k^i(m) = P_r(R_{k+1}^N | d_k = i, S_k = m). \tag{13}$$

Substituting (12) and (13) in (11) we obtain:

$$\lambda_k^i(m) = \frac{\alpha_k^i(m) \beta_k^i(m)}{P_r(R_1^N)}. \tag{14}$$

This result can be used to evaluate (7) as

$$L(d_k) = \log \frac{\sum_m \alpha_k^1(m) \beta_k^1(m)}{\sum_m \alpha_k^0(m) \beta_k^0(m)}, \tag{15}$$

where the summations are over all 2^v states.

2.1 Derivation of α

We want to show that (12) can be recursively calculated. We can express (12) as:

$$\begin{aligned} \alpha_k^i(m) &= P_r(d_k = i, S_k = m, R_1^{k-1}, R_k) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(d_k = i, d_{k-1} = j, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(d_{k-1} = j, S_{k-1} = m', R_1^{k-1}) \\ &\quad \times P_r(d_k = i, S_k = m, R_k | d_{k-1} = j, S_{k-1} = m', R_1^{k-1}) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(d_{k-1} = j, S_{k-1} = m', R_1^{k-1}) \\ &\quad \times P_r(d_k = i, S_k = m, R_k | d_{k-1} = j, S_{k-1} = m'), \end{aligned} \tag{16}$$

since d_{k-1} and S_{k-1} completely define the path at time k–1, the received information R_1^{k-1} is irrelevant. We let

$$\gamma_{i,j}(R_k, m, m') = P_r(d_k = i, S_k = m, R_k | d_{k-1} = j, S_{k-1} = m'). \tag{17}$$

Substituting (12) and (17) into (16) we obtain the iterative equation

$$\alpha_k^i(m) = \sum_{m'} \sum_{j=0}^{1} \alpha_{k-1}^j(m') \gamma_{i,j}(R_k, m, m'). \tag{18}$$

2.2 Derivation of β

In a similar way we can recursively calculate the probability $\beta_k(m)$ from the probability $\beta_{k+1}(m)$. Note that this is possible only after the whole block of data is received. Relation (13) becomes:

$$\begin{aligned} \beta_k^i(m) &= P_r(R_{k+1}, R_{k+2}^N | d_k = i, S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(d_{k+1} = j, S_{k+1} = m', R_{k+1}, R_{k+2}^N | d_k = i, S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(R_{k+2}^N | d_{k+1} = j, S_{k+1} = m', R_{k+1}, d_k = i, S_k = m) \\ &\quad \times P_r(d_{k+1} = j, S_{k+1} = m', R_{k+1} | d_k = i, S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} P_r(R_{k+2}^N | d_{k+1} = j, S_{k+1} = m') \\ &\quad \times P_r(d_{k+1} = j, S_{k+1} = m', R_{k+1} | d_k = i, S_k = m), \end{aligned} \tag{19}$$

since $d_{k+1} = j$ and $S_{k+1} = m'$ completely define the path at time k+1, making R_{k+1} irrelevant. Also (19) is valid only for those $d_{k+1} = j$ and $S_{k+1} = m'$ that extend from the path $d_k = i$ and $S_k = m$, allowing these conditions to be dropped.

Substituting (13) and (17) into (19) we obtain the iterative equation

$$\beta_k^i(m) = \sum_{m'} \sum_{j=0}^{1} \beta_{k+1}^j(m') \gamma_{j,i}(R_{k+1}, m', m). \tag{20}$$

2.3 Derivation of $\gamma_{i,j}(R_k, m, m')$ and $\gamma_{j,i}(R_{k+1}, m', m)$

The probabilities $\gamma_{i,j}(R_k, m, m')$ and $\gamma_{j,i}(R_{k+1}, m', m)$ can be determined from the transition probabilities of the discrete Gaussian memoryless channel and transition probabilities of the encoder trellis. From (17) and using Bayes' rule we have

$$\begin{aligned} \gamma_{i,j}(R_k, m, m') &= P_r(R_k | d_k = i, S_k = m, d_{k-1} = j, S_{k-1} = m') \\ &\quad \times P_r(d_k = i | S_k = m, d_{k-1} = j, S_{k-1} = m') \\ &\quad \times P_r(S_k = m | d_{k-1} = j, S_{k-1} = m'). \end{aligned} \tag{21}$$

Each of the terms in (21) can be expressed as follows. We have

$$\begin{aligned} &P_r(R_k | d_k = i, S_k = m, d_{k-1} = j, S_{k-1} = m') \\ &\quad = P_r(R_k | d_k = i, S_k = m), \end{aligned} \tag{22}$$

since R_k is dependent only on the path at time k, not on the path at time k–1. This is the probability of the received data for a particular path at time k. We also have

$$\begin{aligned} &P_r(d_k = i | S_k = m, d_{k-1} = j, S_{k-1} = m') \\ &\quad = P_r(d_k = i) = 1/2, \end{aligned} \tag{23}$$

since d_k is independent of the current state S_k or path at time k–1 and each possible d_k are equally likely. Finally we have

$$P_r(S_k = m|d_{k-1} = j, S_{k-1} = m') = \begin{cases} 1 \ ; m' = S_b^j(m), \\ 0 \ ; m' \neq S_b^j(m), \end{cases} \quad (24)$$

where $S_b^j(m)$ is the previous state given the path defined by $S_k = m$ and $d_{k-1} = j$ (this will be a unique path since the code is systematic). Substituting (22) to (24) into (21) we obtain

$$\gamma_{i,j}(R_k, m, m') = \begin{cases} \delta_i(R_k, m) \ ; \ m' = S_b^j(m), \\ 0 \quad\quad ; \ m' \neq S_b^j(m), \end{cases} \quad (25)$$

where

$$\delta_i(R_k, m) = P_r(R_k|d_k = i, S_k = m)/2. \quad (26)$$

Substituting (25) into (18) the summation over all m' will disappear and the only surviving m' will be $S_b^j(m)$. Thus, (18) becomes

$$\alpha_k^i(m) = \delta_i(R_k, m) \sum_{j=0}^{1} \alpha_{k-1}^j(S_b^j(m)). \quad (27)$$

A more intuitive graphical representation of (27) is given in Figure 1.

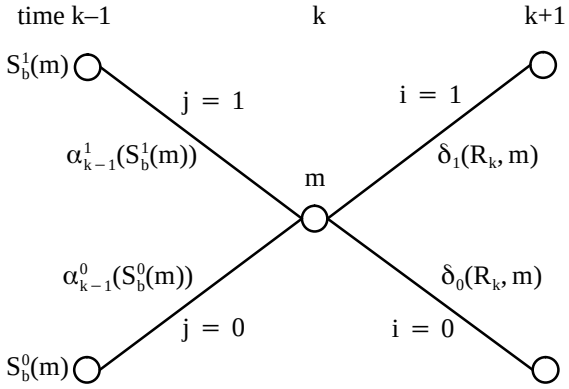

Figure 1: Graphical representation of (27).

Looking at $\gamma_{j,i}(R_{k+1}, m', m)$, we have

$$\begin{aligned} \gamma_{j,i}(R_{k+1}, m', m) &= P_r(R_{k+1}|d_{k+1} = j, S_{k+1} = m', d_k = i, S_k = m) \\ &\quad \times P_r(d_{k+1} = j|S_{k+1} = m', d_k = i, S_k = m) \\ &\quad \times P_r(S_{k+1} = m'|d_k = i, S_k = m) \\ &= P_r(R_{k+1}|d_{k+1} = j, S_{k+1} = m')P_r(S_{k+1} = m'|d_k = i, S_k = m)/2 \\ &= \begin{cases} \delta_j(R_{k+1}, m') \ ; \ m' = S_f^i(m), \\ 0 \quad\quad ; \ m' \neq S_f^i(m), \end{cases} \end{aligned} \quad (28)$$

where $S_f^i(m)$ is the next state given the path defined by $S_k = m$ and $d_k = i$. Substituting (28) into (20) the summation over all m' will disappear and the only surviving m' will be $S_f^i(m)$. Thus, (20) becomes

$$\beta_k^i(m) = \sum_{j=0}^{1} \beta_{k+1}^j(S_f^i(m))\delta_j(R_{k+1}, S_f^i(m)). \quad (29)$$

A graphical representation of (29) is given in Figure 2.

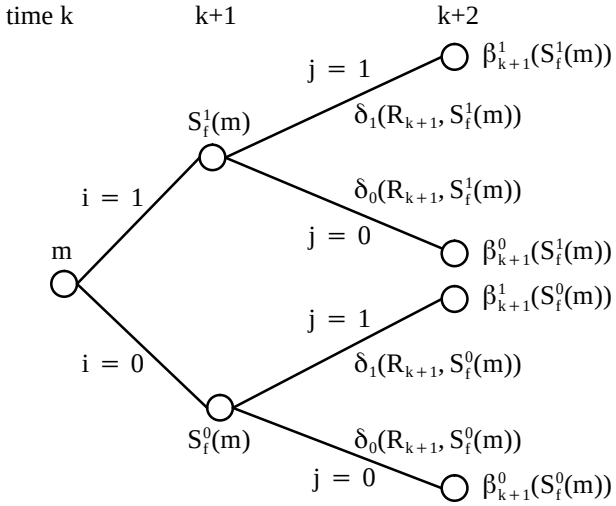

Figure 2: Graphical representation of (29).

2.4 Redefinition of δ

Since p_k and q_k are independent, we rearrange (26) to

$$\delta_i(R_k, m) = P_r(x_k|d_k = i, S_k = m)P_r(y_k|d_k = i, S_k = m)/2, \quad (30)$$

which for an AWGN channel with mean zero and variance σ^2 becomes

$$\begin{aligned} \delta_i(R_k, m) &= \frac{1}{2\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x_k - (2d_k - 1))^2\right) dx_k \\ &\quad \times \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(y_k - (2Y_k - 1))^2\right) dy_k \\ &= K_k \exp\left(\frac{2}{\sigma^2}(x_k i + y_k Y_k^i(m))\right), \end{aligned} \quad (31)$$

where K_k is a constant and dx_k and dy_k are the differentials of x_k and y_k, respectively. Also, we rewrite d_k as i and Y_k as $Y_k^i(m)$ to underline that Y_k is a function of the input bit $d_k = i$ and the encoder state $S_k = m$. Since the constant K_k in (31) doesn't affect $L(d_k)$ in (15), we redefine (31) as

$$\delta_i(R_k, m) = \exp\left(\frac{2}{\sigma^2}(x_k i + y_k Y_k^i(m))\right). \quad (32)$$

Note that (32) inherently changes the definition of (12) and (13) by some constant value. However, this is not important since $L(d_k)$ remains the same regardless.

2.5 The algorithm

With α, β, and δ defined, the steps of the decoding algorithm are

1) Starting at time k = 1, compute $\delta_i(R_k, m)$ for all received symbols and store in an array of size $2^n N$ (for the 2^n possible coded symbols, in this case n = 2).

2) Initialise $\beta_N^i(S_b^i(0)) = 1$ for i = 0, 1 and $\beta_N^i(m) = 0$ for all other m and i. Starting at time k = N–1, iteratively compute $\beta_k^i(m)$ using (29) and store in an array of size $2^v N$ (since $\beta_k^1(m) = \beta_k^0(m')$ where $S_f^1(m) = S_f^0(m')$ we can reduce the array size by half).

3) Initialise $\alpha_1^i(0) = \delta_i(R_1, 0)$ for i = 0, 1 and $\alpha_1^i(m) = 0$ for all $m \neq 0$ and i = 0, 1. Starting at time k = 2, iteratively

compute $\alpha_k^i(m)$ using (27). For each k (including k = 1) compute $L(d_k)$ using (15) and output $\hat{d}_k$.

When computing $\alpha_k^i(m)$ and $\beta_k^i(m)$ we can renormalise the values to prevent them from becoming too large or too small. For example, if α_{min} and α_{max} are the smallest and largest values of $\alpha_k^i(m)$, we can divide each $\alpha_k^i(m)$ by

$$\alpha_{mid} = \exp((\log \alpha_{min} + \log \alpha_{max})/2), \tag{33}$$

to ensure we don't over or underflow $\alpha_k^i(m)$. To reduce the number of computations, this renormalisation should be applied to $\delta_i(R_{k+1}, m)$. A similar renormalisation can also be applied to $\beta_k^i(m)$.

This algorithm appears to have been independently discovered in [3]. The equations are expressed in a different form, but we believe that the algorithms are inherently the same. The definitions of α and β in [3] have a complicated denominator which can be eliminated since they are constant in value. With this simplification, the algorithms become essentially the same.

For continuously encoded data, it is impractical to store all the δ and β values. Like the Viterbi algorithm, a finite length need only be stored. For example, the storage length could be 2N. At time 2N–1 we let $\beta_{2N-1}^i(m) = 1$ for all m and i. The β's and α's then iteratively calculated as in the algorithm, except we only output $\hat{d}_k$ from time k = 0 to N–1. The process then repeats with all the time indices incremented by N. The total decoder delay will be 4N. The number of computations also increases since we have to compute each $\beta_k^i(m)$ twice.

With the simplified algorithm, it is now possible to implement a hardware *maximum a posteriori* (MAP) decoder (which is another name for this algorithm). To simplify the algorithm we express $L(d_k)$ from (15) as

$$L(d_k) = \mathop{\mathrm{E}}_{m} A_k^1(m) + B_k^1(m) - \mathop{\mathrm{E}}_{m} A_k^0(m) + B_k^0(m), \tag{34}$$

where we define

$$x \mathrel{E} y = \log(e^x + e^y), \tag{35}$$

$$\begin{aligned} \mathop{\mathrm{E}}_{m=0}^{2^v-1} f(m) &= f(0) \mathrel{E} f(1) \mathrel{E} \cdots \mathrel{E} f(2^v - 1) \\ &= \log\left(\sum_{m=0}^{2^v-1} \exp(f(m))\right), \end{aligned} \tag{36}$$

and taking the log of (27), (29), and (32) we have

$$A_k^i(m) = D_i(R_k, m) + \mathop{\mathrm{E}}_{j=0}^{1} A_{k-1}^j(S_b^j(m)), \tag{37}$$

$$B_k^i(m) = \mathop{\mathrm{E}}_{j=0}^{1} B_{k+1}^j(S_f^i(m)) + D_j(R_{k+1}, S_f^i(m)), \tag{38}$$

$$D_i(R_k, m) = \frac{2}{\sigma^2}(x_k i + y_k Y_k^i(m)). \tag{39}$$

The calculation of (38) is almost the same as the add, compare–select (ACS) function of the Viterbi algorithm, except that the E function replaces the compare–select function. It is also possible to rearrange the order of computation of (37) so that it is similar to (38), except that we are going in the forward direction (as in the Viterbi algorithm). $A_k^i(m)$ and $B_k^i(m)$ can be thought of as "state metrics" and $D_i(R_k, m)$ as the "branch metrics" of the algorithm. Taking the log of (33) the renormalisation metric becomes

$$A_{mid} = (A_{min} + A_{max})/2, \tag{40}$$

where A_{min} and A_{max} are the smallest and largest values of $A_k^i(m)$. Renormalisation is performed by subtraction. Two's complement arithmetic can be used for the addition and subtracting functions. The E function can be implemented using look–up–tables, although the number of bits used to represent each state metric will be limited.

The MAP decoder will now have $3 \times 2^{v+1}$ additions and $2^{v+2} - 2$ E computations per decoded bit (assuming the decoder stores all the received samples and ignoring branch metric calculations). The Viterbi algorithm has 2^{v+1} additions and $2^v - 1$ compare–select computations. If we consider that the E function is equivalent to the compare–select function in complexity (for a discrete implementation this is true since the same number of chips are used) then the MAP algorithm is about four times as complex as the Viterbi algorithm.

2.6 Results

The performance of the MAP algorithm compared with the Viterbi and Li et. al. [4] algorithms is given in Table 1. The four state systematic convolutional code with polynomials $(5,7)_8$ was used. The simulations were performed with at least 1500 bit errors in each case.

Table 1: BER for a four state code.

E_b/N_0 (dB)	0.0 10^{-2}	1.0 10^{-2}	2.0 10^{-2}	3.0 10^{-3}	4.0 10^{-4}
Viterbi	8.29	4.19	1.57	4.33	9.50
Li et. al.	7.93	4.09	1.56	4.26	9.48
MAP	7.81	4.03	1.53	4.27	9.36

The worst performance is given by the Viterbi algorithm, with a slightly improved performance by the Li et. al. algorithm. As expected, the MAP algorithm gave the best results (although only slightly). Although the improvements were very small, this may be important for Turbo–codes. In the first stages of decoding, obtaining even a slight improvement with very noisy data could lead to much greater improvements in the latter stages of decoding. Also, the MAP algorithm inherently provides soft outputs which can be used in the next stage of decoding.

3 APPLICATION IN ITERATIVE DECODING

The definition of turbo–codes was introduced in [2] and the principles of iterative decoding are clearly described in [3]. We will not repeat the derivations here, but only show how (32) will change for iterative decoding. Figure 3 presents a generic turbo–encoder.

ENC^- is a rate half systematic encoder in the "horizontal" dimension and $ENC^|$ is the second encoder in the "vertical" dimension. The interleaver block (INT) changes the input order of the information bits d_k for the second encoder. The

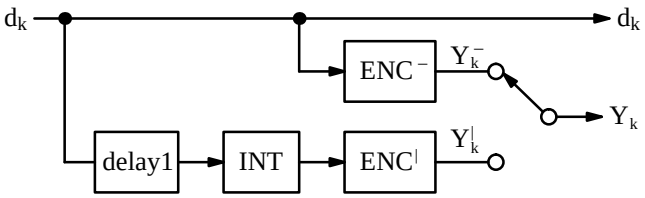

Figure 3: Turbo encoder.

switch alternatively selects one of the coded bits produced by the two systematic encoders. The pair (d_k, Y_k) is the output of the Turbo encoder which is BPSK or QPSK modulated and sent through an AWGN channel. The receiver structure is presented in Figure 4. The DELAY1 block introduces a delay equivalent to the delay due to the decoder DEC$^-$. For the first iteration, the extrinsic information, $z_k^|$ is zero. The whole delay of the decoder is Δ.

As described in [3], $L(d_k)$ for DEC$^-$ can be expressed as follows

$$L(d_k) = x_k + z_k^| + z_k^-. \tag{41}$$

Note that we have x_k and y_k prescaled by $2/\sigma^2$ before analogue to digital conversion. Also, $z_k^|$ is the a priori extrinsic value for d_k and z_k^- is the extrinsic information generated by DEC$^-$ for d_k. Thus, for DEC$^-$ we redefine $\delta_i(R_k, m)$ as

$$\delta_i(R_k, m) = \exp\big((x_k + z_k^|)i + y_k Y_k^i(m)\big). \tag{42}$$

3.1 Results

Simulations were performed for the 16 state rate half systematic convolutional code given in [2] (with polynomials $(21,37)_8$) and a 20×20 block interleaver of 400 bits at an E_b/N_0 of 2.0 dB (the lowered code rate of 0.49 due to forcing the start and end states to 0 is taken into account). Table 2 gives the performance of the Turbo decoder at each iteration stage (up to eight) at an E_b/N_0 of 2.0 dB. A total of 10^7 information bits were simulated. The final result is very close to that found in [3].

Table 2: Turbo decoder performance at E_b/N_0 = 2.0 dB.

Iter.	1	2	3	4	5	6	7	8
BER 10^{-4}	172	15.0	4.13	2.49	1.88	1.73	1.67	1.55

4 CONCLUSIONS

This paper presented a simplification of the modified Bahl algorithm which can be used in an iterative decoder. Its complexity is approximately four times the complexity of the Viterbi decoder allowing the algorithm to be efficiently implemented in hardware. The algorithm can now be used to perform more simulations of Turbo–codes and to verify the high performance claimed in [2].

5 REFERENCES

1. Bahl, L., Cocke, J., Jelinek, F., and Raviv, J., "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inform. Theory, vol. IT–20, pp. 284–287, Mar. 1974.

2. Berrou, C., Glavieux, A., and Thitimajshima, P., "Near Shannon limit error–correcting coding and decoding: Turbo–Codes," ICC'93, Geneva, Switzerland, pp. 1064–1070, May 1993.

3. Hagenauer, J., Robertson, P., and Papke, L., "Iterative (Turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms," ITG Conf., Frankfurt, Germany, Oct. 1994.

4. Li, Y., Vucetic, B., Sato, Y., and Furuya, Y., "A soft–output Viterbi algorithm," 1st Int. Mobile and Personal Commun. Systems Workshop, Adelaide, SA, pp. 223–231, Nov. 1992.

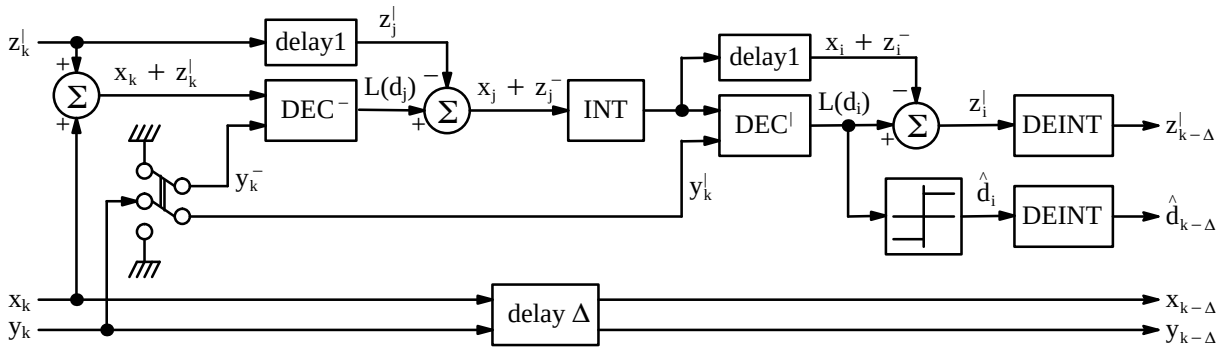

Figure 4: Iterative Turbo Decoder.