On Symbol-based Turbo Codes for cdma2000

F. Khaleghi, A. Khandani, N. Secord, and A. Gutierrez
Nortel Networks
Email: farideh@nortel.ca
Tel: (613) 765-1256

Abstract

Turbo codes have enjoyed a great attention in recent years. Among different soft-output decoding algorithms of Turbo codes only maximum a posteriori (MAP) decoding as an iterative soft-output decoder (BCJR) allows for achieving an acceptable BER performance at Eb/No levels within only 1 dB of the value corresponding to the Shannon capacity.

The drawback of the MAP algorithm is the excessive memory required. Here, we investigate the so-called symbol-based Turbo codes. These codes allow for a reduced required memory by 30% for the BCJR method .

1. Introduction

Turbo-codes have replaced the convolutional codes in cdma2000 (third generation of IS-95 system) RTT proposal submitted to the ITU in June 1998 as a candidate RTT for IMT-2000. The input bits to the Turbo code encoder are divided to the blocks of a certain length depending on the data rate as specified in the cdma2000 RTT [1]. In a so-called parallel Turbo code, the information bits are sent as the systematic bits and then the block of the information bits is encoded by the first constituent encoder to produce the first parity bits and the bits of the original block after being interleaved are encoded by the second constituent encoder producing the second set of the parity bits. The systematic and parity bits are then transmitted in a serial fashion. The interleaver in the conventional Turbo codes is based on a bit-by-bit interleaving. In symbol-based Turbo codes [2], the structure of a traditional Turbo code is modified to act over a sub-block of input bits where the interleaver is constructed to permute the sub-blocks (while preserving the relative order of the bits within each sub-block). The resulting codes are called "Symbol-based Turbo-codes" [2]. By parsing the input data block into n-bit symbols, we are in essence merging n sections of the encoder trellis into one.

As known for the MAP Turbo decoding, the entire state metric history must be stored in the memory during the forward recursion up to the end of the trellis, at which point the backward algorithm begins and decisions can be made starting with the last branch, with a need to store only the last set of the state metrics computed backward. This memory requirement is obviously excessive. The merging of n trellis stages results in a reduction in the effective block length by 1/n. A shorter effective block length translates into fewer stages for the forward and backward recursions, and consequently, less values need to be stored.

An important special case of the above code structure is obtained when the length of the sub-blocks is equal to two bits. For this special case (n=2), first the advantages are outlined in section 2. In section 3, it is shown that the computation complexity of the Turbo decoder remains roughly the same while the required memory size is reduced by more than 30%. Some simulation results are presented in section 4, and finally the conclusions are given.

2. Advantages of the Symbol-Based Turbo Codes

In a Symbol-based Turbo encoder, the Turbo interleaver operates on groups of bits. The two constituent convolutional encoders are the same and still operate on a bit-by-bit basis.

In the decoder, the APP values of the symbols are passed from one iteration to the next as opposed to the APP values of the bits. This reduces the number of the stages of the trellis and hence the number of the state probabilities. The assumption of the independence of the conditional bit probabilities in subsequent stages of the trellis is removed. The assumption of the independence is not completely valid. The corresponding dependency is stronger for bits which are closer to each other. As a result, the performance of the symbol-based decoding algorithm will be closer to that of a true maximum likelihood decoding.

Therefore, the benefits of the symbol-based Turbo codes can be summarized as; 1) reduction in the Turbo decoding memory requirements, and 2) reduction in Turbo interleaver complexity.

2.1. Turbo Decoder Memory Requirement Benefits

The number of trellis stages in a Symbol-based Turbo-code is half of the standard Turbo-code. This reduces the size of the RAM required to store the state probabilities by a factor of two,

0-7803-5668-3/99/$10.00 © 1999 IEEE.

resulting in 33% overall reduction in the RAM size for a BCJR decoder.

In a Turbo-code of block length N with three streams of output and 8 states, the size (in words) of RAM required to store the state probabilities is equal to 8 N. This is 2/3 of the total RAM. The other 1/3 is used to store the channel outputs (3 N) and the LLR values (N). Using Symbol-based Turbo-code, the memory for storing the state probabilities is reduced by a factor of two, resulting in an overall saving of 4/12 = 33% in the RAM size.

In section 3, it is shown that the computational complexity of the Symbol-based Turbo decoder is practically identical to that of the conventional bit Turbo decoder.

2.1.1. Symbol-Based Turbo Codes and Sliding Window Algorithm

Sliding-window is a method to reduce the memory requirement of a Turbo-code decoder (at the price of an increase in the computational complexity [3,4]).

The key idea in sliding window decoding is to repeat the back-ward recursion in overlapping windows to avoid storing the state probabilities for the entire block. Each computational window is composed of two consecutive sub-windows of lengths A and B where, (i) the backward computations performed in sub-window B are redundant and are used to initialize the recursion in sub-window A, and (ii) the state probabilities are stored only in sub-window A. The price to pay is a relative increase of (B/A) in the computational complexity of the backward recursion. If the size of window A is very small compared to the block size N, then the reduction in the required memory size is not significant. The required RAM for a bit-level Turbo decoder consists of storage space for the channel outputs (3 N) (i.e., corresponding to frame size), LLR values (N), and state probabilities (8 A, where A is the window size). The RAM size is reduced to 4 A for the state probabilities of the Symbol-based Turbo decoder.

2.2. Turbo Interleaver Complexity Reduction

The interleaving in Turbo-codes is achieved either by generating the interleaver on the "fly," or by storing the interleaver table. In the first approach, the interleaver structure is stored in a parametric form and the corresponding memory requirement is negligible. There is however a price to pay in terms of the computational complexity. In the second approach, the entire interleaver is stored, requiring a large RAM size. Using symbol-based Turbo-code with n = 2 results in a reduction in the corresponding computational (or memory) requirement by a factor of two. Performance of Symbol-based Turbo-code: According to the extensive simulations conducted, symbol-based Turbo-code with n = 2 results in a similar (or even better) performance as compared to a traditional bit-by-bit interleaving.

3. Decoding Complexity of Symbol Based Turbo Codes

In this section we show that the computation complexity of both the bit-based and symbol-based Turbo decoders are comparable. In the following analysis, we have not included the complexity reduction of the interleaving and de-interleaving operations that are part of the decoding process. We consider a decoding algorithm for Turbo-codes (with M = 3 memory elements, 2 M = 8 states). Decoding is performed in the log-domain, where the bit probabilities are stored as the log-probability values. These values are normalized such that the log-probability of bit 0 is equal to zero (this normalization is achieved by subtracting the log-probability of the zero from the log-probability of one). Such a normalization reduces the memory required to store the APP bit probabilities (to one memory location for each bit) and also simplifies the subsequent calculations (additions involving zero values are avoided).

The state probabilities are also stored as log-probabilities where the values are normalized to have zero value for the zero state (for both forward recursion, α values, and back-ward recursion, β values). This normalization is simply achieved by subtracting the log-probability of the zero state from the other states at each stage of the trellis. This normalization results in the following advantages:

- It reduces the memory used to store the state probabilities to 7/8 of its original value. This is because the probabilities of zeros do not need to be stored.
- It simplifies the subsequent calculations (which will largely compensate for the computational complexity associated with the normalization operation).
- It reduces the chance of overflow throughout the forward and backward recursions.

We assume that at each stage of the trellis (in the forward or backward recursion) the branch metrics (γ values) are pre-computed (for all the

branches of the current stage) and the results are subsequently used to perform the rest of the computations related to that stage. This results in a reduction in complexity because there exist several branches with the same label in each section of the trellis for which we would like to avoid the recalculation of the γ values.

We consider the conventional approach to decoding in which the trellis is processed one stage at a time (n = 1), and also an alternative in which two subsequent stages of the trellis are processed at once (n = 2);"Symbol-based Turbo-code".

In the following, we provide a description of the calculations involved in the decoding algorithm and provide a comparison between the two cases of n = 1 (conventional decoding) and n = 2.

3.1. Operation I: Calculation of Branch Metrics (γ values)

The following figure shows two stages of a trellis diagram corresponding to an 8-state Turbo Code. The notation a/ab on the branch labels specifies the soft values of the encoder input and output, respectively, where a is the encoder input and ab is the encoder output (systematic and parity bits, respectively).

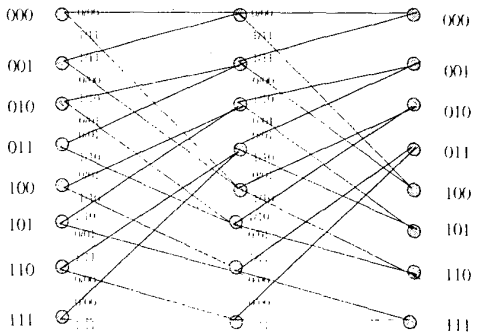

Figure 1. 8-state Trellis Diagram

The values involved in the computation of the branch metrics for n = 1 are as shown in Table 1. Since the log-probabilities are normalized such that the log-probability of zero is zero, the log-probabilities of zero are shown as zero. The log-probability of one is a non-zero number and is shown by x, y, and z. For the first column, the first element is the APP value of the corresponding bit (in the log domain) and the second and the third elements are the conditional probabilities of the corresponding systematic and parity bits as received at the channel output. In the second column is the number of computations required to generate the corresponding values.

Table 1. Branch Metric Computation Values

n=1 APP and Conditional Probabilities	Computations
000	
00z	
xy0	1
xyz	1
Total	2

One can compute all the entries of this table using two operations per trellis stage (one addition for xy and one extra addition for xyz). This operation is repeated in the forward and backward recursions resulting in a total of 4 operations per trellis stage.

Table 2 shows the values for n = 2, where the first two entries are the APP values of the corresponding bits (in the log domain). The second two entries are the conditional probabilities of the corresponding systematic bits (in the log domain) and the last two entries are the conditional probabilities of the corresponding parity bits (in the log domain):

Table 2. n=2 Values

000000	00000f	0000e0	0000ef
0b0d00	0b0d0f	0b0de0	0b0def
a0c000	a0c00f	a0c0e0	a0c0ef
abcd00	abcd0f	abcde0	abcdef

One can compute all the entries of this table using 12 operations per trellis stage as shown in the following table:

Table 3. Computations for n = 2

a 0 c 0	1
0 b 0 d	1
0 b 0 d 0 f	1
0 b 0 d e 0	1
0 b 0 d e f	1
a 0 c 0 0 f	1
a 0 c 0 e 0	1
a 0 c 0 e f	1
a b c d 0 0	1
a b c d 0 f	1
a b c d e 0	1
a b c d e f	1
Total	12

It should be mentioned that these operations are repeated two times, once during the forward and once during the backward recursions. This leaves us with a total of 4, and 24 operations per trellis stage for the cases of n = 1, n = 2, respectively.

This results in a net value of 24 – 4 = 20 operations, or in other words a savings of 10 operations per trellis stage for the symbol based Turbo Code with n = 2 when compared to n = 1.

3.2. Operation II: Calculation of Forward State Probabilities (α values)

Figure 2 shows the basic unit of computations involved in updating the α values for n = 1 and n = 2.

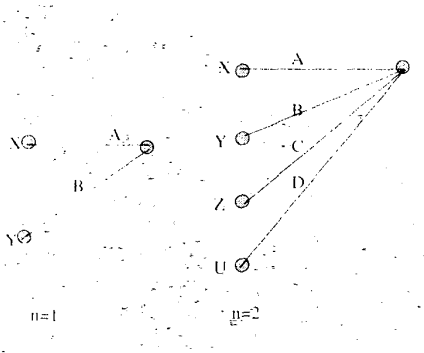

Figure 2. Computations for Updating Alpha Values

The corresponding computations for n = 1, n = 2 are summarized in the following tables, where I(·), L(·) denote the "inverse-log" and the "log" look-up table operations.

Table 4. n = 1 alpha Computations

X+A	1
Y+B	1
I(X+A)	1
I(Y+B)	1
I(X+A)+I(Y+B)	1
L(I(X+A)+I(Y+B))	1
Total	6

Table 5. n = 2 alpha Computations

X+A	1
Y+B	1
Z+C	1
U+D	1
I(X+A)	1
I(Y+B)	1
I(Z+C)	1
I(U+D)	1
I(X+A)+I(Y+B)+I(Z+C)+I(U+D)	1
L(I(X+A)+I(Y+B)+I(Z+C)+I(U+D))	1
Total	12

This means that the complexity of these operations for n = 1 and n = 2 is the same (total of 6x 8 states = 48 operations per trellis stage). A similar argument holds for the backward recursion (total of 6 x 8 states = 48 operation per trellis stage). Note that due to the normalization applied to the state probabilities, we need only to include branches starting from a non-zero state in the above computations. The corresponding saving in the complexity is part of what we referred to in the earlier discussion concerning the advantages obtained through normalization of the state probabilities.

3.3. Operation III: Normalization of the state probabilities

This operation takes 14 operations in each stage of the trellis (7 operations to normalize the α values and 7 operation to normalize the β values). For n = 2, this will reduce to 14 operation per two stage of the trellis, meaning 7 operations per stage to the advantage of n = 2 vs. n = 1.

3.4. Operation IV: Mixing the α, β, γ values and updating the bit probabilities

The following tables show the basic unit of computations involved in mixing the results for the α, β and γ values. Noting that the total number of branches in each section of the trellis for n = 1, n = 2 is equal to 16, 32, respectively, we conclude that the complexities for the two cases are the same. The results of these computations are subsequently added together (after taking out of the log domain) to give the bit probabilities.

Table 6. Computations for n = 1

$\alpha+\beta+\gamma$	2*16=32
I($\alpha+\beta+\gamma$)	16
Σ I ($\alpha+\beta+\gamma$)	2*8=16
L (Σ I ($\alpha+\beta+\gamma$))	2
Normalization of log-probabilities	1
Total	**67**

Table 7. Computations for n = 2

$\alpha+\beta+\gamma$	2*32=64
I($\alpha+\beta+\gamma$)	32
Σ I ($\alpha+\beta+\gamma$)	(4*8) =32

L (Σ I (α+β+γ)) Normalization of log-probabilities	4 4
Total	**136**

This results in 2 operations per trellis stage in favor of n = 1 vs. n = 2.

3.5. Comparison of the computational complexity per trellis stage for n=1 & n=2

A summary of calculations are shown in the following table.

Table 8. Comparison of n = 1 and n = 2 calculations

	n=1	n=2
Operation I	4	24
Operation II	96	96
Operation III	14	7
Operation IV	67	68
Total	**181**	**195**

It is observed that the computational complexity per trellis stage of the two methods are almost the same, while the symbol-based method results in a saving in the size of the RAM memory required to store the state probabilities by a factor of two.

4. Simulation Results

In this section, we present some simulation results on the performance of the symbol-based Turbo codes. Figure 2 depicts the performance results for block length of 288 for a code rate of ¼.

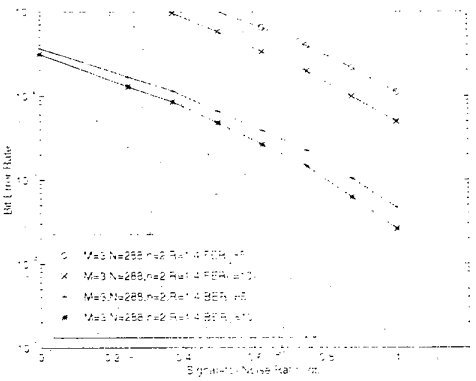

Figure 2. BER and FER performance for N=288, AWGN channel

In Figure 3, we compare the performance of the 2-D linear congruential sequence (LCS) symbol-based interleaver (IS-2000 Turbo Interleaver) with that of the corresponding bit-based 2-D LCS interleaver. These results also indicate that no performance penalties are associated with the symbol-based Turbo codes.

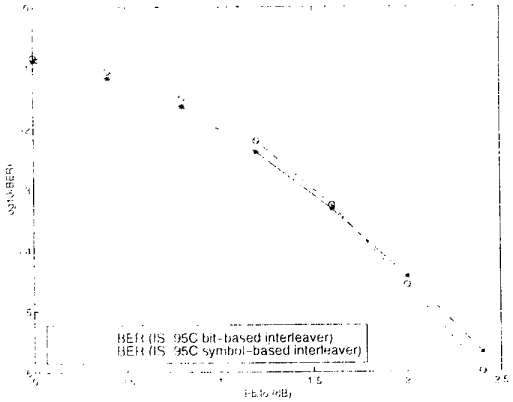

Figure 3. BER and FER performance for N=192, AWGN channel

5. Conclusions

In summary, the benefits provided by the Symbol Based Turbo code are twofold – reduction in memory for decoding, and reduction in Turbo Interleaver complexity. The decoding memory is reduced by 33% for the BCJR decoding and may be reduced some for Window decoding as compared to conventional Turbo Codes, when the size of the window is large. It was shown that the decoding complexity is approximately the same for Symbol Based Turbo codes as compared to conventional Turbo Codes. Finally, the computational requirement for the Turbo Interleaver is reduced by a factor of two when compared to that of conventional Turbo Codes.

References

[1] cdma2000 RTT, Draft Text for "95c" Physical Layer (Revision 4).

[2] M. S. Bingeman and A. K. Khandani,"Symbol-based Turbo-codes", to appear in the IEEE Communication Letters (also refer to: "Symbol-based Turbo-codes for Wireless Communications", M. S. Bingeman, M.A.Sc. Thesis, University of Waterloo).

[3] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Soft-output decoding algorithms in iterative decoding of Turbo-codes", TDA Progress Report 42-124

[4] A. J. Viterbi, "An institutive justification and a simplified implementation of the MAP decoder for convolutional codes", pp. 260-264,