Implementation issues of 3rd generation mobile communication turbo decoding

Theorem 7 *For any $d \geq 1$ and reliability set $\mathcal{R}$, it holds that*

$$d^*_{\mathbf{OE}}(d,[0,1]) \leq d^*_{\mathbf{OE}}(d,\mathcal{R}) \leq d^*_{\mathbf{OE}}(d,\{1\}) = 2\lceil d/2\rceil \tag{20}$$

where equality holds in the first inequality (and thus $d^*_{\mathbf{OE}}(d,\mathcal{R})$ equals (18) with $\mu = 0$) if $d \leq 2$ and $\bar{\mathcal{R}} \supseteq \{0,1\}$, or $d \geq 3$ and $d \not\equiv 1(4)$ and $\bar{\mathcal{R}} \supseteq \{1/2,1\}$, or $d \geq 3$ and $d \equiv 1(4)$ and $\bar{\mathcal{R}} \supseteq \{1/3,2/3,1\}$.

V Discussion

Forney's original GMD decoder [2] achieves the full Hamming distance d of the code in (at most) $\lceil d/2\rceil$ trials. Kovalev [3] has shown that when limiting the maximum number of trials, still a considerable fraction of the Hamming distance can be exploited. For single-trial decoding, as under consideration in this paper, this fraction is at least $2/3$ for the simple fixed and threshold erasing strategies, and at least $3/4$ for the somewhat more complex optimized erasing strategy. Explicit expressions for the realizable distance by single-trial GMD decoders for important classes of reliability sets have been provided in Sections II-IV. Due to lack of space, proofs have been omitted. These will appear in [5], as $l = 1$ cases of more general results on l-trial GMD decoding. For fixed erasing, the results from Theorems 1 and 2 can be extended to explicit expressions for any value of l [4].

References

[1] I.M. Boyarinov, "Method of decoding direct sums of products of codes and its applications," *Problemy Peredachi Inform.*, vol. 17, no. 2, pp. 39-51, 1981.

[2] G.D. Forney, Jr., "Generalized minimum distance decoding," *IEEE Trans. Inform. Theory*, vol. 12, pp. 125-131, April 1966.

[3] S.I. Kovalev, "Two classes of minimum generalized distance decoding algorithms," *Problemy Peredachi Inform.*, vol. 22, no. 3, pp. 35-42, 1986.

[4] J.H. Weber and K.A.S. Abdel-Ghaffar, "Limited-trial generalized minimum distance decoding with fixed erasing", *to appear in* Proceedings IEEE International Symposium on Information Theory, Sorrento, Italy, June 25-30, 2000.

[5] J.H. Weber and K.A.S. Abdel-Ghaffar, "Limited-trial generalized minimum distance decoding," *in preparation, to be submitted to IEEE Trans. Inform. Theory.*

[6] V.V. Zyablov, "Optimization of concatenated decoding algorithms," *Problemy Peredachi Inform.*, vol. 9, no. 1, pp. 26-32, 1973.

IMPLEMENTATION ISSUES OF 3RD GENERATION MOBILE COMMUNICATION TURBO DECODING

J. Dielissen

Technical University Eindhoven,
Information and Communication
Systems, Department of Electrical
Engineering

J. Huisken

Embedded Systems Architectures
on Silicon (ESAS)

Philips Research Laboratories*
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
dielisse@natlab.research.philips.com, jos.huisken@philips.com

Turbo Decoding is part of the standard for 3rd generation mobile communication. There are several different implementation techniques for the Soft-Input Soft-Output decoder of a Turbo Decoder.
This paper explains four implementation techniques, their performances and implementation costs.

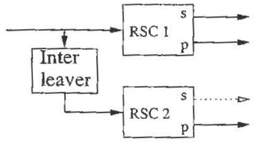
INTRODUCTION

The introduction of 'turbo codes' by Berrou et.al. [1] in 1993 opened up new perspectives in channel coding theory. The outstanding bit error rate performances and the wide range of applications created a large interest in this coding scheme. Due to recent development of mobile communication and the continuing miniaturisation of integrated circuits it is possible to integrate a Turbo Decoder in mobile communication applications [4],[6].
For making this integration cost-effective we have to reduce the implementation costs. This reduction is investigated by exploring the design space between performance and implementation cost. Since the Soft-Input Soft-Output decoder is the scalable part of a Turbo Decoder this will be the subject of investigation in this paper. We restrict ourselves to Turbo Decoding for 3GPP, which is explained in the next section. Implementation techniques are explained and their performance and costs analysed in the succeeding sections. By costs we mean implementation costs in terms of silicon area and power dissipation.

* corresponding address

TURBO CODING FOR 3GPP

The Turbo coding scheme for 3rd Generation Partnership Project (3GPP)[1] in Frequency Division Duplex (FDD) mode[2] consists of two 8-state Recursive Systematic Coders (RSC). Figure 1 shows the structure of the encoder. The transfer function of both RSCs is stated in equation 1. The output of the Turbo Encoder consists of the original message (systematic output from RSC 1) and the parity output from both RSCs, resulting in a rate $\frac{1}{3}$ code. This structure is known under the name Parallel Concatenated Convolution Code (PCCC).



$$G(D) = [s,p] = \left[1, \frac{1+D^1+D^3}{1+D^2+D^3}\right] \qquad (1)$$

Figure 1: encoder

The block-length of the message is determined by the interleaving length. In 3GPP this interleaving length is specified between 40 and 5114 symbols. The input symbols are written row wise into a matrix. After inter- and intra- row permutations the matrix is read column wise. For terminating an 8-state code 3 additional trellis-steps are required, resulting in 6 additional bits for terminating each RSC (3 systematic and 3 parity bits). The total number of bits which are sent over the channel is 3*B+12, where B is the block-length of the message.

The 3GPP standard only specifies the encoder, resulting in some algorithmic freedom for the decoder. In the next section several alternatives are used to implement an efficient Turbo Decoder.

IMPLEMENTATION ISSUES OF TURBO DECODING

The Turbo Decoder consists of two modules: the (de-)interleaver and the Soft-Input Soft-Output (SISO) module. In this article we focus on the SISO module, because of the possibility to exchange performance with implementation cost. For SISO decoding there are two families: BCJR (named after its inventors Bahl, Cocke, Jelinek, and Raviv) and SOVA (Soft Output Viterbi Algorithm). BCJR-type algorithms are more expensive, have better performance and allow easy scalability. Due to these last two arguments we choose to exploit BCJR-type algorithms further. For example when using a BCJR-type SISO module, which has better performance than SOVA-type algorithms, we can reduce the average number of iterations [6] resulting in lower implementation costs.

[1]http://www.3GPP.org

[2]3GPP TSG RAN WG1:"TS 25.212 Multiplexing and Channel Coding (FDD) V3.1.1 (1999-12)"

Within the BCJR family there are two interesting algorithms: Max-log-MAP and Max$\star$-log-MAP [5]. Max-log-MAP is a simplification of Max$\star$-log-MAP with respect to a correction term, which is applied in Max$\star$-log-MAP. The performance gain of scaling extrinsic information in the Max-log-MAP algorithm [2], brings Max-log-MAP very close to Max$\star$-log-MAP.

IMPLEMENTATION TECHNIQUES FOR BCJR-TYPE SISO DECODERS

There are several different implementation techniques for the BCJR-type algorithms [3]. In this section the main four techniques are explained. The first two techniques are true BCJR-type algorithms, the last two represent approximations of these algorithms. For each implementation technique numerous different calculation orders are possible. For the basic calculation order we can start at the beginning of the trellis, at the end of the trellis, or simultaneously at the beginning and end of the trellis.

We first start with a full block implementation technique. It's schedule is visualised in Figure 2. Forward and backward recursion are carried out over the full block of data. The backward recursion is started after the last forward recursion. The soft output calculation starts immediately after the calculation of the corresponding backward state metric vector in the backward recursion. The corresponding forward state metric vector is retrieved from the memory in which it was saved during forward recursion.

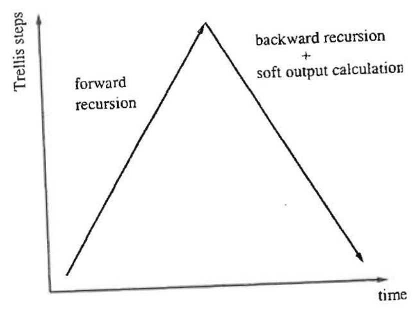


Figure 2: Full block BCJR

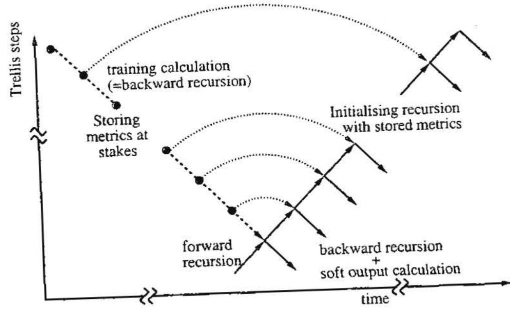

Trellis steps
training calculation (=backward recursion)
Storing metrics at stakes
Initialising recursion with stored metrics
forward recursion
backward recursion + soft output calculation
time

Figure 3: Efficient Implementation-BCJR

The schedule of the second true BCJR-type algorithm, Efficient Implementation-BCJR (Figure 3) starts with the backward recursion, saving the state metric vectors at trellis-steps B, B-W, . . . , W, further referred to as stakes. When the forward calculation reaches a stake, the backward recursion is initialised with the corresponding backward recursion stake. Soft output calculations start immediately after the calculation of the corresponding backward recursion state metric vector. This true BCJR-type algorithm is named for its efficient state metric vector memory, which is scaled by minimal the square root of the block-length against linear memory requirements of full block BCJR.

Figure 4 shows the schedule of sliding window with training calculations, being the first non-true BCJR-type algorithm. In this technique, the backward recursion is not initialised with the exact state metric vector, but with an approximation. Several steps ahead in the trellis, the training recursion is started with an uniform state metric vector. After several training calculations the state metric vector converges to an estimation of the correct state metric vector. When conducting 5*K training calculations performance loss is negligible for an AWGN channel. K is the constrain length of the code (for 3GPP K=4). For Rayleigh fading channel characteristics 10*K training calculations need to be calculated. Note that training calculations imply both more silicon area and power dissipation.

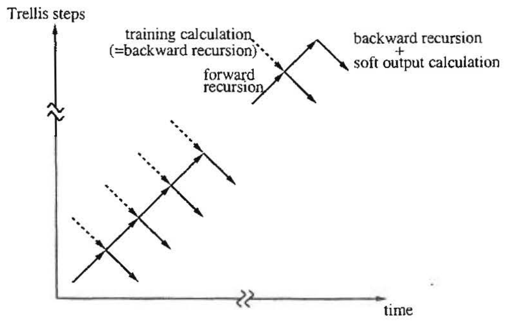

Trellis steps
training calculation (=backward recursion)
forward recursion
backward recursion + soft output calculation
time

Figure 4: Sliding window with training calculations

The Next Iteration Initialisation (NII) implementation technique [3] is shown in Figure 5. The backward recursions at the stakes are initialised with stakes from the previous iteration. Note that SISO 1 can only be initialised with stakes from SISO 1, resulting in twice the amount of stake memory, compared to EI-BCJR. In the first iteration of this technique, either EI-BCJR initialisation can be used, training calculation can be calculated, or an uniform state metric vector can be used for initialisation.

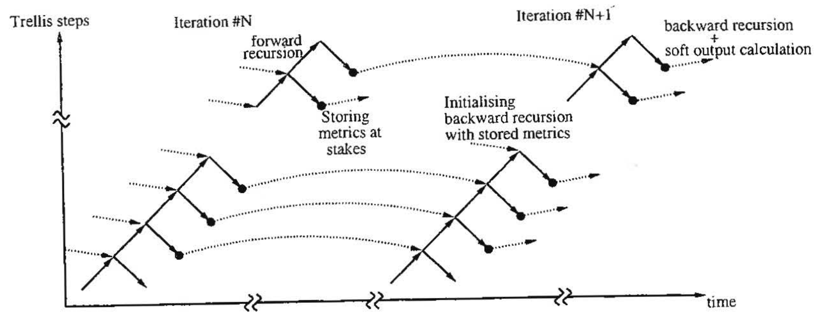

Trellis steps
Iteration #N
Iteration #N+1
forward recursion
backward recursion + soft output calculation
Storing metrics at stakes
Initialising backward recursion with stored metrics
time

Figure 5: Sliding window Next Iteration Initialisation

RESULTS

We start with the reference simulation. Figure 6 shows the simulation results of the Max⋆-log-MAP algorithm using the full block implementation technique and the inter-leaver as explained before.

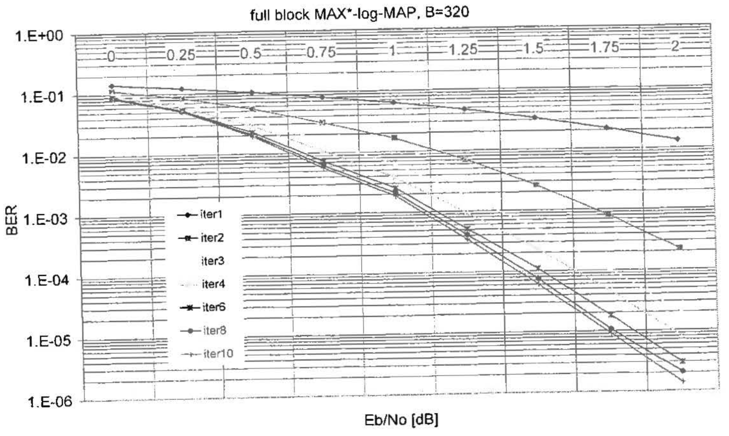


Figure 6: BER versus Eb/No for different iterations

Since EI-Max⋆-log-MAP has the same performance and the performance degradation of sliding window Max⋆-log-MAP with training calculations is negligible, they are not shown in this figure. All simulations are in floating point precision and use an AWGN channel model. Max⋆ operations use a 64 entry linear distributed [0, 0.1, . . . , 6.3] lookup table, for their correction term ($max\star(x,y) \approx max(x,y) + f_c(|x-y|)$).

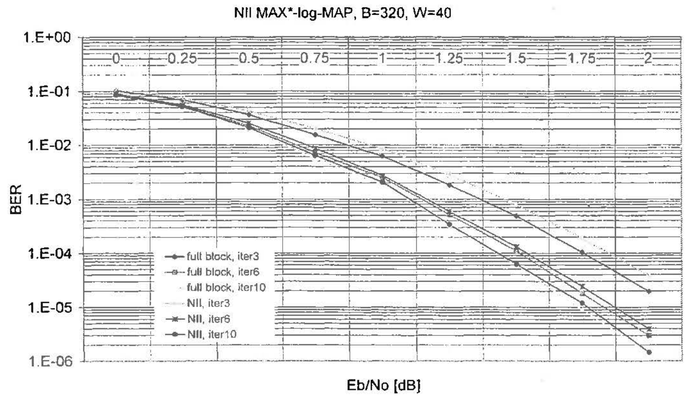


Figure 7: BER versus Eb/No showing performance degradation of NII schedule

Figure 7 shows the performance degradation of the Next Iteration Initialisation (NII) algorithm. In the first iteration uniform state metric vectors are filled in at the stakes. The performance differences become negligible for higher iteration numbers. Figure 8 gives a clearer view of the performance differences in the first iterations. In the first iteration a large error is made. Every following half iteration the performance differences are reduced. Using initialisation schemes which have lower bit error rates in the first iteration is only interesting if the implementation costs are less than a quarter of the cost of an iteration.

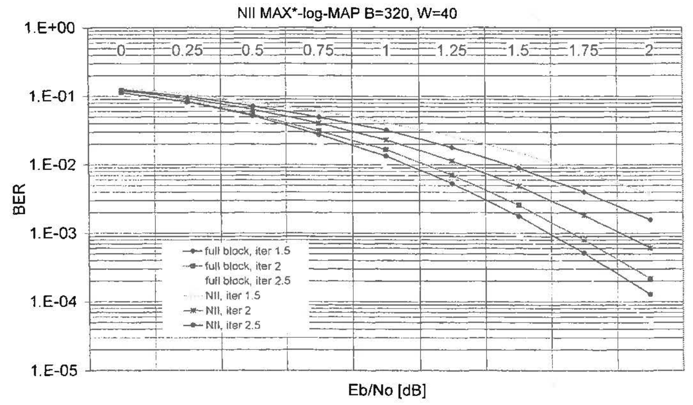


Figure 8: BER versus Eb/No showing performance degradation in first iterations

We implemented the basic SISO module used in the explained techniques: a sliding window Max-log-MAP decoding algorithm on a variable window-length. Each forward recursion stake is passed to the next window, while the backward recursion stakes are provided externally. From this basic architecture all explained implementation techniques can be derived.

About 40% of area costs of the basic SISO decoder consists of local memory. Local memory consists of temporary saved branch metrics, state metric vectors, and stakes. For memory we use static RAM. The memory costs for a Turbo Decoder are:

costs	basic solutions	training calculations	NII
LLR	$B^*(P_a+3P_l)$	$B^*(P_a+3P_l)$	$B^*(P_a+3P_l)$
Branch metrics	$W^*(P_{sys}+P_{par})$	$2^*W^*(P_{sys}+P_{par})$	$W^*(P_{sys}+P_{par})$
State metric vectors	$W^*N^*P_{state}$	$W^*N^*P_{state}$	$W^*N^*P_{state}$
Stake vectors			$2^*(B/W)^*N^*P_{state}$

B = block-length (= 40 . . . 5120)	W = window-length (= 40)
P_{sys} = systematic LLR (= 7 bit)	P_{par} = parity LLR (= 4 bit)
P_{state} = state metric (= 8 bit)	P_a = A-priori information (= 6 bit)
P_l = intrinsic soft input (= 4 bit)	N = number of states (= 8)

Figure 9 shows the implementation costs of the SISO module and the LLR memories (intrinsic and A-priori information). The implementation costs of sliding windows with training steps is constant, NII grows with the square root of the block-length and the LLR memory costs on turbo decoding level grow approximately linear with the block-length. All implementation cost figures are for a $0.18\mu m$ CMOS technology. Implementation costs of the interleaving is not taken into account.

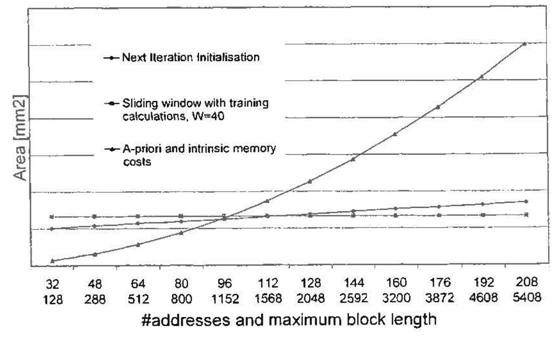


Figure 9: Turbo Decoder implementation costs

DISCUSSION

In this article several different implementation techniques are shown. Memories dominate the chip area. On a chip for 3GPP these memories might be reused for other functions. It is therefore not trivial to choose one implementation technique. Simulations show that increasing the window length in NII results in higher performance in the first iterations. When it is possible to freely choose the window-length, a trade-off exist between the average number of iterations (influenced by the performance) and the latency of the SISO decoder (influenced by the window-length). Low latency might avoid expensive buffering in front of the Turbo Decoder and low average iteration numbers result in reduced power consumption.

REFERENCES

[1] C. Berrou, A. Glavieux, and P Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. In *IEEE Proceedings of ICC '93*, pages 1064–1070, May 1993.

[2] S. Crozier, Ken Gracie, and Andrew Hunt. Efficient turbo decoding techniques. Technical report, Communications research Centre, 3701 Carling Avenue, P.O. Box 11490, Station H Ottawa, Ontario, 1999.

[3] A. Dingninou, F. Raouafi, and C. Berrou. Organisation de la memoire dans un turbo decodeur utilisant l'algorithm sub-map. In *Proceedings of Gretsi*, pages 71–74, September 1999. France.

[4] G. Masera, G. Piccinini, M. Ruo Roch, and M. Zamboni. VLSI architectures for turbo codes. *IEEE Transactions on VLSI Systems*, 7(3):369–378, September 1999.

[5] P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and suboptimal map decoding algoritms operating in the log domain. In *Proceedings 1995 International Conference on Communications*, pages 1009–1013, 1995.

[6] Z. Wang, H. Suzuki, and K. K. Parhi. VLSI implementation issues of turbo decoder design for wireless applications. In *IEEE Workshop on Signal Processing Systems*, pages 503–512, 1999.

BOX-FUNCTIONS AND MISMATCHED LOG-LIKELIHOOD RATIOS

A.J.E.M. Janssen & A.G.C. Koppelaar*

Philips Research Laboratories WY-81,82
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
Email : {a.j.e.m.janssen,arie.koppelaar}@philips.com

The inputs and outputs of an a-posteriori probability (APP) decoder can be represented as log-likelihood ratios. The box-function is an explicit input-output relation of an APP decoder. We introduce the log-likelihood ratio property and verify that the box-function exhibits this property. Furthermore, we study the effect of mismatched inputs to the box-function.

1 INTRODUCTION

The good performance of iterative decoding of Turbo Codes [1] has stimulated the research of parallel and serial concatenated coding schemes. Key elements in iterative decoders are the soft-input soft-output decoders and the interleaver. A symbol-by-symbol a-posteriori probability (APP) decoder is an optimal soft-input soft-output decoder. Such a decoder can work both in the probability domain (APP decoder) and in the log-likelihood ratio (LLR) domain (log-APP decoder).

Working in the LLR domain, the soft outputs of an optimal decoder will be represented as LLR's, provided that the soft inputs are also represented as LLR's. Furthermore, in an iterative decoding scheme, the component decoders assume that the systematic input and the a-priori input (supplied by the other component decoder) are uncorrelated. Depending on the size of the interleaver and the spreading characteristics of the interleaver this assumption stands shorter or longer in the course of iterating. We have identified three reasons why an iterative decoder (e.g. working in the LLR domain) has inferior performance compared to an ideal iterative decoder :

- the inputs do not correspond to LLR's.
- the inputs are correlated.

* Communicating author