LOW COMPLEXITY ITERATIVE DECODING OF VARIABLE-LENGTH CODES

Catherine Lamy[*]

THALES Communications France
catherine.lamy@fr.thalesgroup.com

Lisa Perros-Meilhac[*]

New Logic Technologies
Sophia-Antipolis, France

ABSTRACT

Widely used in data compression schemes, where they allow to reduce the length of the transmitted bistream, variable-length codes (VLCs) are very sensitive to channel noise. In fact, when some bits are altered by the channel, synchronisation losses can occur at the receiver, possibly leading to dramatic symbol error rates. The key point is to appropriately use the residual source redundancy at the decoding side, for instance by considering it as an implicit channel protection that can be exploited to provide error correction capability. We propose in this paper a soft-input soft-output low complexity VLC decoding algorithm. Combined with a convolutional code SISO decoder, this new Soft Output Stack Algorithm (SOSA) is used in an iterative joint decoder and simulations results over an Additive White Gaussian Noise (AWGN) channel are shown.

1. INTRODUCTION

Providing highly compressed sources that make the transmission bandwidth-efficient, VLCs are widely used in data compression schemes even though they generate bitstreams very sensitive to channel perturbations. To cope with this phenomenon, some introduced modified VLCs such as self-synchronising Huffman codes [1], reversible VLCs [2], proposed to re-introduce redundancy in the bitstream by inserting an error correcting code [3] in the chain or by conjuging error detection capacity and arithmetic coding [4].

As shown by several authors [3, 5], the key point is to appropriately use the residual source redundancy at the decoding side. This redundancy can be considered as an implicit channel protection by the decoder, and be exploited as such to provide error correction capability of the variable-length coded source. The optimal VLC decoder acts, then, as an estimator of the transmitted information by selecting the sequence according to the Maximum A Posteriori (MAP) criterion. The improvement achieved when compared to classical hard decoding is significant [3, 6, 7, 8, 9], and recent works showed that low-complexity approximate MAP decoders could be used, that provide performance similar to all existing soft decoding algorithms while exhibiting a complexity close to the hard decoding case [10].

We propose in this paper a SISO low complexity VLC decoding algorithm relying on the hard output sequential algorithm presented in [10]. This new Soft Output Stack Algorithm (SOSA) is then combined with a classical convolutional code SISO in an iterative joint decoder, following the approaches introduced in [11, 12, 13].

This paper is organised as follows. Section 2 introduces the variable-length codes model and describes the VLC tree structure. In Section 3 is proposed a soft-output version of the stack algorithm given in [10], with two possible soft outputs generation methods. Performance for this new stack algorithm are then given in Section 4, where it is inserted in a joint iterative decoder for concatenated convolutional and variable-length codes. Simulation results are given and finally some conclusions are drawn.

[*]The authors were with Philips Recherche France, Suresnes, France.

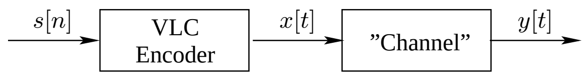
2. NOTATIONS AND MODEL FOR SOFT-INPUT SOFT-OUTPUT VLC DECODING ALGORITHMS

Let us consider the system model presented in Figure 1, where the VLC is assumed to be binary and tree structured. A symbol sequence of length R, $\mathbf{s}[1:R]$, is mapped to a binary sequence of length T, $\mathbf{x}[1:T]$, by following the rule edicted by the VLC table of size K. This sequence is BPSK modulated, transmitted over an AWGN channel and demodulated at the receiver side. The received sequence, $\mathbf{y}[1:T]$, is decoded by a soft input VLC decoder which computes a soft output in the form of the log *a posteriori* ratio $\Lambda(x[t])$:

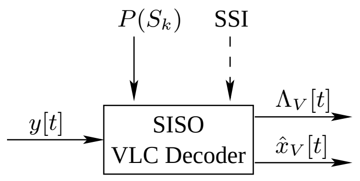
$$\Lambda(x[t]) = \log \frac{P(x[t]=1|\mathbf{y}[1:T])}{P(x[t]=0|\mathbf{y}[1:T])} \quad \text{for } 1 \le t \le T, \quad (1)$$

and a hard bit estimate sequence, $\hat{x}[1:T]$ or a hard symbol estimate sequence $\hat{s}[1:R]$.

As every SISO decoder, the VLC SISO decoder will derive the reliability information on bits $\Lambda(x[t])$ by taking into account any *a priori* knowledge it has on the VLC encoder. In our case, the *a priori* knowledge will consist of the VLC tree structure, the occurrence probabilities of the symbols and any other source side information, such as possibly the concerned sequence number of symbols.



(a) VLC encoding process model



(b) SISO VLC decoder model

Fig. 1. VLC encoding and decoding system model.

Let describe the VLC tree with the following notations, as illustrated in Figure 2 for code $\mathcal{C}_{11}$ defined by Table 1:

- S_j: the j^{th} symbol in the VLC table, $1 \leq j \leq K$,
- I_j: the j^{th} intermediate node, $1 \leq j \leq L$,
- $\mathcal{N}_p, = \{I_1, \ldots, I_L, S_1, \ldots, S_K\}$, the set of nodes having a predecessor,
- $p_t(l)$ ($l \in \mathcal{N}_p$): the *a priori* probability of the branch reaching the node l at time t.

	Probability	$\mathcal{C}_{11}$
S_1	0.33	00
S_2	0.30	01
S_3	0.18	11
S_4	0.10	100
S_5	0.09	101
Average length		2.19

Table 1. VLC code $\mathcal{C}_{11}$, taken from [14].

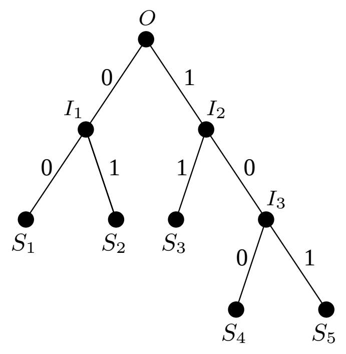


Fig. 2. VLC tree representation.

3. A SOFT-INPUT SOFT-OUTPUT STACK ALGORITHM

Even though MAP algorithms provide very good results in terms of error correction, they happen to be fairly complex, as they imply to look through the whole tree in both ways (forward and backward). Stack algorithms on the other hand are much simpler and yet can reach similar performance, as shown in [10][15]. They are however primarily symbol-level decoding algorithms, hence must be adapted to provide reliability information on bits.

Our new algorithm proceeds in two main steps: first, decoding part estimates the transmitted VLC sequence, and second a post-processing derives the soft values. The considered post-processing operation reveals itself much less complex than the one proposed in [16] and provides soft values more relevant than those obtained using the post-processing proposed in [17] for an equivalent complexity.

3.1. A bit level VLC stack decoder

The first step in the algorithm consists in applying a classical stack algorithm [18] on the "huge" tree formed by concatenating several times the considered Huffman tree until the number of bits and the number of symbols of the considered sequence is reached.

As sequential VLC decoding algorithms compare bit sequences of different lengths, a specific metric must be used. For each node l in the set $\mathcal{N}_p$, we define the metric associated to the branch leading to this node at time t by adapting the metric proposed in [10]:

$$m(l, y[t]) = -\log P(y[t]|v(l)) - \log p_t(l) + \log P_0(y[t]) \quad (2)$$

where the three terms appearing in the metric stand respectively for the symbol likelihood at node l, the *a priori* probability on the considered branch and the metric P_0 introduced by Fano and Massey for sequential variable length decoding. In practice, the term $p_t(l)$ is for simplicity reasons be approximated by the *a priori* probability of the branch $p(l)$, which can be directly obtained as explained in [19] from the tree representation of the VLC table and the codeword probabilities which are assumed to be known by the decoder.
The algorithm different steps are as follows:

1. Create the *tree* by defining relations between *nodes* computing the *a priori* probability $p(l)$ associated with each branch.
2. Initialize the stack by placing the initial node (O) with metric 0 in the stack.
3. Compute the metric of the succeeding branches of last node of the top path. If an extended path reaches a symbol node, increment the number of symbols associated to this path. Delete the top path from the stack and insert the extended paths in the stack.

4. Select the new top path *i.e.* the path of the stack having the smallest cumulative metric. If the last node of this path corresponds to a symbol node, use the origin node O as new starting state.
5. Test if stop conditions are verified: the top path contains the number of bits and the number of symbols of the original sequence. If stop conditions are verified, continue to step 6. Otherwise, return to step 3.
6. Derive soft output values from cumulative metrics of considered paths in the previous processing and stored in the stack.

3.2. Soft outputs generation

Once the sequential decoding process is finished, the post-processing can take place to generate soft outputs. These soft outputs will be derived from the paths that have been examined and stored by the stack decoding algorithm.

Let $\{P_1, \dots, P_r\}$ denote the set of r examined paths stored in the stack. A given path P_i ($1 \le i \le r$) is characterised by a length in bit T_{P_i}, a cumulative metric μ_{P_i} and a sequence of bits $\{\hat{x}_{P_i}[1], \dots, \hat{x}_{P_i}[T_{P_i}]\}$. We propose two different possible solutions for deriving the soft outputs.

Solution 1
Inspired by the bidirectional Soft-Output Viterbi Algorithm (SOVA) presented in [20], this first solution consists in approximating the log-likelihood ratio $\Lambda(x[t])$ by

$$\Lambda(x[t]) \simeq \mu(t,0) - \mu(t,1), \tag{3}$$

where $\mu(t,1)$ (resp. $\mu(t,0)$) is the minimum path metric for all the paths in the stack for which the t^{th} estimated bit is 1 (resp. 0). As in the bidirectionnal SOVA, if $P^\star$ is the path selected by the decoding process, then if $\hat{x}_{P^\star}[t] = i$ ($i = \{0,1\}$), we have $\mu(t,i) = \mu_{P^\star}$.

Solution 2
In this second solution, all the metrics in the paths stored by the stack algorithm and not only the best paths for both values 0 and 1 for t^{th} estimated bit are taken into account. The log-likelihood ratio $\Lambda(x[t])$ is hence approximated by

$$\Lambda(x[t]) \simeq \log\Big(\sum_{\substack{1\le i\le r\\ T_{P_i}>=t\\ \hat{x}_{P_i}[t]=1}} e^{-\mu_{P_i}} \Big/ \sum_{\substack{1\le i\le r\\ T_{P_i}>=t\\ \hat{x}_{P_i}[t]=0}} e^{-\mu_{P_i}} \Big). \tag{4}$$

4. ITERATIVE JOINT DECODING FOR CONCATENATED CC AND VLC CODES

4.1. Joint decoder principle

To validate Section 3 algorithms, let us consider the communication system presented in Figure 3. The transmitter consists first of a VLC source which outputs random VLC symbols according to the chosen VLC codewords probabilities. Grouped into packets of R symbols, these symbols are mapped to a T-bit VLC sequence $\mathbf{x}[1:T]$ by the VLC encoder. The size of this resulting sequence in bits is a source side information (SSI) supposed known at the receiver side. Each sequence is then permuted by a pseudo-random interleaver Π. Note that as the packet size is different for each sequence, a new interleaver must be used (and generated) for each packet. The interleaved sequence $\tilde{\mathbf{x}}[1:T]$ is given as input to a systematic convolutional encoder. The coded sequence $\mathbf{v}[1:T*n/k]$ at the output of this CC encoder is punctured in order to attain the wished transmission rate, modulated by a BPSK modulator and transmitted over an AWGN with variance σ^2.

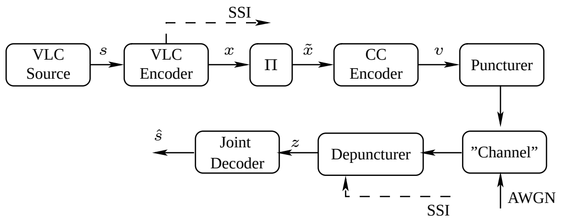


Fig. 3. Transmission chain for joint decoding.

At the receiver side, the sequence is demodulated and depunctured. The depunctured sequence $\mathbf{z}[1:T*n/k]$ is then decoded by the iterative decoder proposed in Figure 4. The iterative decoding process takes place as follows: at the r^{th} iteration, the CC decoder's input consists of the depunctured sequence $\mathbf{z}[1:T*n/k]$ and the *a priori* probabilities ratio $\Phi^{(r-1)}[1:T]$ of the interleaved sequence $\tilde{\mathbf{x}}[1:T]$ obtained at previous iteration. The CC decoder provides the $\tilde{\Lambda}_C^{(r)}[1:T]$ output sequence. At this same r^{th} iteration, the VLC decoder takes as input the observation sequence $\mathbf{y}^{(r)}[1:T]$ derived of the CC decoder output sequence, the *a priori* probabilities of VLC symbols as well as any other available Source Side Information (SSI) and provides the $\Lambda_V^{(r)}[1:T]$ output sequence.

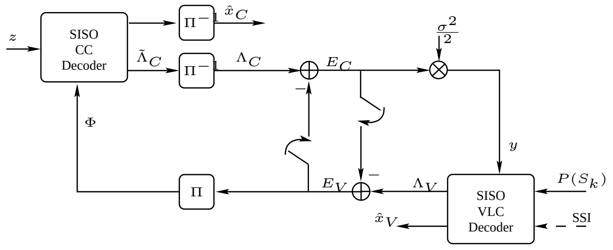


Fig. 4. Proposed iterative decoder.

In order that each decoder takes advantage of the iterative process [21], independent information must be exchanged between the two decoders, that is the so-called *ex-*

trinsic information. We thus define $E_C^{(r)}[t]$ and $E_V^{(r)}[t]$ the extrinsic information about the bit t provided respectively by the CC decoder and by the VLC decoder.

$$\text{For } r \geq 1, \quad E_C^{(r)}[t] = \Lambda_C^{(r)}[t] - E_V^{(r-1)}[t], \tag{5}$$
$$E_V^{(r)}[t] = \Lambda_V^{(r)}[t] - E_C^{(r)}[t], \tag{6}$$

where $E_V^{(O)}[t]$ is set equal to zero.

The CC extrinsic information $E_C^{(r)}[1:T]$ sequence scaled by $\sigma^2/2$ [20] is used as observation for the r^{th} iteration of VLC decoder,

$$y^{(r)}[t] = \frac{\sigma^2}{2} E_C^{(r)}[t]. \tag{7}$$

and the interleaved VLC extrinsic information $\tilde{E}_V^{(r)}[1:T]$ is used as *a priori* probability ratio estimate for the $r+1^{th}$ iteration of the CC decoder,

$$\Phi^{(r)}[t] = \tilde{E}_V^{(r)}[1:T]. \tag{8}$$

4.2. Numerical results

Let us first assess the SOSA algorithm efficiency in the context of short VLCs. Considering that it has perfect knowledge of the number of symbols, we will compare its performance with the one of a maximum *a posteriori* decoding of VLC codes with exact *apriori* knowledge (denoted KMAP algorithm) on the number of symbols by frame. This KMAP algorithm and the two versions SOSA1 and SOSA2 (with respectively solution 1 and 2 for soft outputs derivation) are used to study the performance of concatenated CC+VLC schemes in the joint iterative decoding scheme described in previous Section. In each case, the simulation characteristics follow these guidelines:

- 100 source symbols sequences are generated, and mapped into bits according to the occurence probabilities defined by the VLC table given in Table 2,
- pseudo-random (de-)interleaving is used between the VLC and CC (de)en-coders,
- the optimal distance profile systematic convolutional code CC_A [18] is used: rate $1/2$, generators [40, 73], memory $m = 5$, minimal distance $d = 6$,
- BPSK modulation is used over the AWGN channel.

Considering that the chosen VLC code is not an optimised Huffmann one, but was selected for its minimal Hamming distance [15] equal to 2 between each couple of codewords, a puncturing of rate $R = 8/9$ was applied to allow for fair comparison with standard VLC compression (*e.g.* with the code given in Table 1).
Figure 5 first illustrates the performance of both SOSAs. The two different solutions proposed in Section 3 appear to be roughly equivalent, with perhaps a very small advantage to SOSA2, that was consequently used for the rest of the tests. Figure 6 shows the influence of the stack size in SOSA algorithm, where it can be seen that some gain is achieved

	Probability	Symbols
S_1	0.33	00
S_2	0.30	11
S_3	0.18	010
S_4	0.10	101
S_5	0.09	0110
Average length		2.47

Table 2. short VLC code used in the simulations.

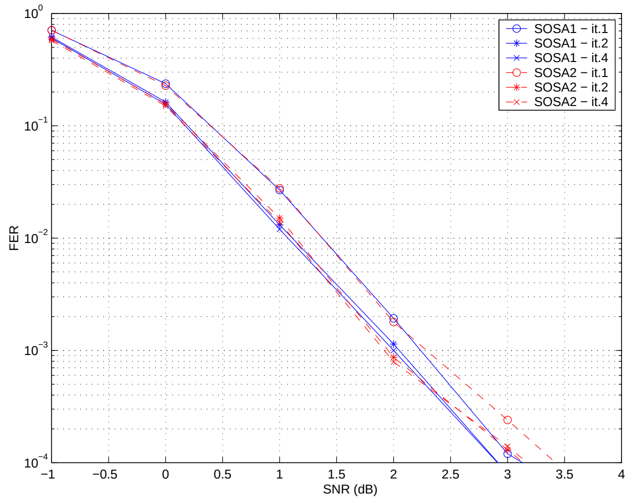


Fig. 5. Performance at VLC decoder output for various iterations with SOSA1 and SOSA2 algorithms (stack size=20).

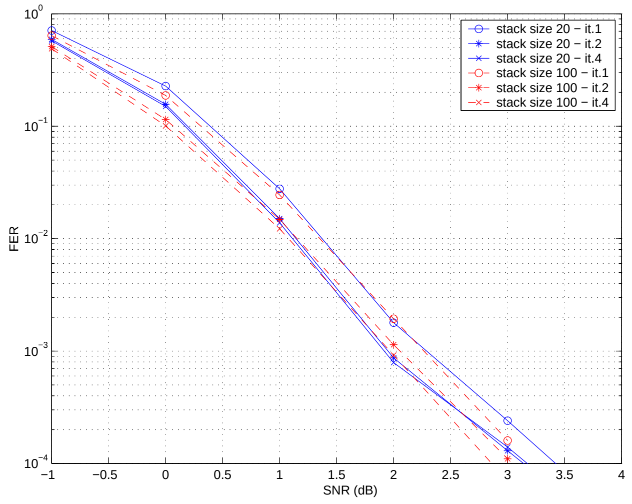


Fig. 6. Performance of the SOSA2 algorithm for various stack sizes and various iterations at the VLC decoder output.

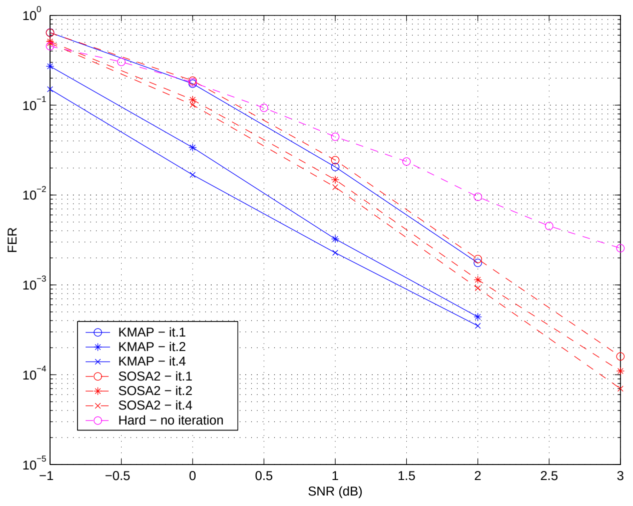


Fig. 7. Performance comparison of KMAP and SOSA2 (stack size=100) algorithms at the VLC decoder output for various iterations.

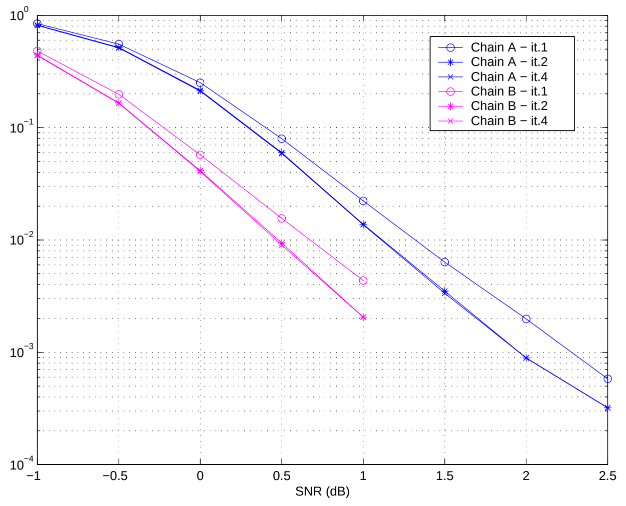


Fig. 8. Performance for vairous iterations and two different CC codes concatenated with VLC_{MPEG-4} code (SOSA2, stack size=20).

when using a stack of maximal size 100 when compared with a stack size of 20. The induced complexity may however be a limiting factor. Finally, Figure 7 shows a performance comparison between the KMAP and the SOSA2 algorithms. As foreseen by previous studies on hard outputs versions of these algorithms [10], both perform similarly for the first iteration. After four iterations, the gain of about 0.8 dB obtained with KMAP algorithm for FER=10^{-3} is not completely reached with SOSA2 algorithm, which only provides about 0.3 dB of gain. Replacing the SISO VLC decoder in the chain by a classical hard VLC decoder (hence without iterating), we find that the gain provided by the iteration process is of about 1.2 dB for FER=2.10^{-3}, which is far from being negligible. Finally, comparing the evolution of the iterations for both KMAP and SOSA2 solutions, and noting that with SOSA2 mainly gains with the second iteration, a possible interpretation may be that the difference may come from the SOSA soft outputs quality.

Let now consider the case of larger VLCs, such as the one used in the MPEG-4 standard, whose statistics are given in Table 3. As a MAP decoding solution complexity is prohibitive for such large codes, only SOSA2 solution is considered here. Figure 8 presents the respective performance of this large VLC code with two different convolutional codes. The first one, denoted by chain A, uses convolutional code CC_A, and the second one, denoted chain B contains the more complex convolutional code CC_B of rate $1/2$, generators [4000, 7154], memory $m = 9$ and minimal distance $d = 8$. Gains of about 0.3 dB can be observed in both cases, and the interest of using a powerful inner code in the serial concatenation is obvious.

Length	Number of symbols	Source statistics
4	2	0.125254
5	2	0.062622
6	6	0.031311
7	6	0.015656
8	20	0.007828
9	17	0.003914
10	27	0.001957
11	44	0.000978
12	28	0.000489
13	24	0.000245
14	32	0.000122
6.5615	Average length	

Table 3. Statistics of the VLC_{MPEG4} code.

5. CONCLUSIONS

A new soft-input soft-output stack VLC decoding algorithm was introduced, that is based on the simplified stack VLC decoding algorithm introduced in [10]. This SISO algorithm allows the use of iterative decoding for concatenated schemes with VLC codes. While much less complex than MAP-based ones, our Soft Output Stack Algorithm (SOSA) was shown to take advantage of the iterative process, both for short and large VLCs, and to be less efficient than MAP-based algorithms by only 0.5 dB in a short VLC context. This small loss may be due to the insufficient quality of the soft outputs generated, which contrarily to the MAP case are by construction not evaluated by taking into account all possible metrics in the tree.

6. REFERENCES

[1] T.J. Ferguson and J.H. Rabinowitz, "Self-Synchronizing Huffman Codes", in *IEEE Trans. on Inform. Theory*, pp. 687–693, vol. 30, n. 4, July 1984.

[2] J. Wen and J.D. Villasenor, "A class of reversible variable length codes for robust image and video coding", in *Proc. of ICIP'97*, pp. 65–68, vol. 2, Santa Barbara, USA, Oct. 1997.

[3] A.H. Murad and T. E. Fuja, "Joint source-channel decoding of variable-length encoded sources", in *Proc. of ITW'98*, pp. 94-95, Killarney, Ireland, June 1998.

[4] C. Boyd, J.G. Cleary, S.A. Irvine, I. Rinsma-Melchert and I.H. Witten, "Integrating error detection into arithmetic coding", in *IEEE Trans. on Commun.*, pp. 1–3, vol. 45, Jan. 1997.

[5] M. Park and D.J. Miller, "Decoding entropy-coded symbols over noisy channels by MAP sequence estimation for asynchronous HMMs", in *Proc. of CISS'98*, pp. 477-482, Princeton, USA, Mar. 1998.

[6] M. Park and D.J. Miller, "Joint source-channel decoding for variable-length encoded data by exact and approximate MAP sequence estimation," *IEEE Trans. on Commun.*, pp. 1–6, vol. 48, n. 11, Jan. 2000.

[7] J. Wen and J.D. Villasenor, "Utilizing soft information in decoding of variable length codes," in *Proc. of DCC'99*, pp. 131–139, Snowbird, USA, Mar. 1999.

[8] S. Kaiser and M. Bystrom, "Soft decoding of variable-length codes," in *Proc. of ICC'00*, pp. 1203–1207, vol. 3, 2000.

[9] C. Lamy and O. Pothier, "Reduced complexity Maximum A Posteriori decoding of variable-length codes," in *Proc. of Globecom'01*, pp. 1410–1413, vol. 2, San Antonio, USA, Nov. 2001.

[10] L. Perros-Meilhac and C. Lamy, "Huffman tree based metric derivation for a low-complexity sequential soft VLC decoding," in *Proc. of ICC'02*, pp. 783–787, vol. 2, New York, USA, Apr.-May 2002.

[11] R. Bauer and J. Hagenauer, "Symbol-by-symbol MAP decoding of variable length codes," in *Proc. of CSCC'00*, pp. 111–116, München, Germany, Jan. 2000.

[12] K.P. Subbalakshmi and J. Vaisey, "On the joint source-channel decoding of variable-length encoded sources," in *IEEE Trans. on Commun.*, vol. 49, n. 12, pp. 2052–2055, Dec. 2001.

[13] K. Lakovic and J. Villasenor, "Combining variable-length codes and turbo codes," in *Proc. VTC'02*, vol. 4, pp. 1719–1723, Birmingham, USA, May 2002.

[14] R. Bauer and J. Hagenauer, "Iterative source/channel-decoding using reversible variable length codes," in *Proc. of DCC'00*, pp. 93–102, Snowbird, USA, Mar. 2000.

[15] A.J. Viterbi and J.O. Omura, *Principles of Digital Communication and Coding.* McGraw-Hill Book Company, New York, 1979.

[16] R. Sivasankaran and S.W. McLaughlin, "Twin-stack decoding of recursive systematic convolutional codes," in *IEEE Trans. on Commun.*, pp. 1158–1167, vol. 49, n. 7, July 2001.

[17] G. Lanzetta, H.A. Cabral and D.J. Costello, "Soft-input/soft-output sequential decoding," in *Proc. of ISITA'00*, Honolulu, USA, Nov. 2000.

[18] S. Lin and D.J. Costello, *Error control coding: fundamentals and applications*, Prentice-Hall, Englewood Cliffs, 1983.

[19] L. Guivarch, J.-C. Carlach, and P. Siohan, "Joint source-channel soft decoding of Huffman codes with turbo-codes," in *Proc. of DCC'00*, pp. 83–91, Snowbird, USA, Mar. 2000.

[20] B. Vucetic and J. Yuan, *Turbo codes: Principles and Applications.* Kluwer Academic Publishers, 2000.

[21] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," in *IEEE Trans. on Commun.*, pp. 1261–1271, vol. 44, Oct. 1996.