ON EFFICIENT SOFT-INPUT SOFT-OUTPUT ENCODING OF CONVOLUTIONAL CODES

Andreas Winkelbauer and Gerald Matz

Institute of Communications and Radio-Frequency Engineering, Vienna University of Technology
Gußhausstraße 25//389, 1040 Vienna, Austria; email: {andreas.winkelbauer, gerald.matz}@nt.tuwien.ac.at

ABSTRACT

We study efficient algorithms for soft-input soft-output (SISO) encoding of convolutional codes. While the BCJR algorithm has been suggested for SISO encoding, we show that a forward recursion on the code's trellis is sufficient to compute the a posteriori probabilities of the code bits. We further propose a shift-register based SISO encoding algorithm for non-recursive convolutional encoders, whose complexity scales linearly with constraint length and block length. Finally, we assess the complexity of the proposed algorithms and we discuss approximations to facilitate practical implementation.

Index Terms— Soft information processing, channel coding, convolutional codes

1. INTRODUCTION

Soft information processing is a key ingredient of many advanced receiver concepts we know today. The ubiquity of *soft-input soft-output* (SISO) processing is due to pioneering work, e.g., by Viterbi [1], Bahl et al. [2], and Hagenauer [3, 4]. This work has paved the way for turbo codes [5] and numerous iterative receiver algorithms which have revolutionized communication theory since 1993.

So far, SISO processing has been mostly applied at the receiver side of communication systems, whereas it has received little attention in the context of transmit processing. This motivated us to study the principles and efficient implementations of SISO channel *encoding* based on convolutional codes. SISO encoding is useful whenever the data to be encoded is not known exactly but only in terms of probabilities. There are a number of possible applications where the extension of conventional *hard-input hard-output* (HIHO) encoding to SISO encoding is beneficial. Examples include, but are not limited to, encoding of data from noisy finite-alphabet sources, joint source-channel encoding, distributed coding in wireless relay networks, soft re-encoding in cooperative scenarios [6, 7], and soft network coding.

SISO encoding using the BCJR (MAP) algorithm [2] has been proposed in the context of wireless relay networks [6, 7]. In this paper, we develop SISO encoders whose computational complexity and memory requirements are significantly lower than BCJR-based implementations. For the special case of non-recursive convolutional encoders, we prove that the corresponding SISO encoder can be implemented in a very intuitive and appealing way using shift registers. We also provide a detailed comparison of the computational complexity and the memory requirements of all SISO encoding algorithms. Finally we discuss approximations that further reduce the complexity of SISO encoding and facilitate practical implementation.

The rest of the paper is organized as follows. In Section 2, the problem of SISO encoding using convolutional codes and the existing BCJR-based implementation is described. We propose more efficient algorithms for SISO encoding in Section 3 and we analyze their computational complexity and memory requirements in Section 4. Conclusions are provided in Section 5.

Funding by FWF Grant N10606 (SISE-Infonet).

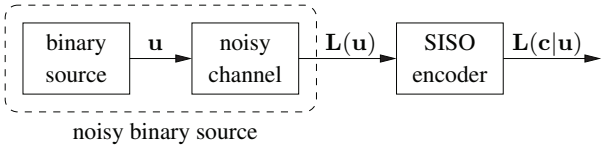

Fig. 1. Block diagram for the SISO encoding of a noisy source.

2. BACKGROUND

Our basic system model for SISO encoding is shown in Fig. 1. Here, the soft inputs are modeled as output of a noisy binary source. This model is general enough to cover most of the applications mentioned above. The source emits a sequence $\mathbf{u} = (u_0\ u_1\ \dots\ u_{K-1})^T$ of K independent information bits. These bits are transmitted over a memoryless noisy channel. We assume that the output of the channel consists of a sequence $\mathbf{L}(\mathbf{u}) = (L(u_0)\ L(u_1)\ \dots\ L(u_{K-1}))^T$ of log-likelihood ratios (LLRs) for the source bits $\mathbf{u}$ (this assumption results in no loss of generality since the LLRs are sufficient statistics for the bits). The LLRs constitute the input to the SISO encoder and are related to the bit probabilities as

$$L(u_k) \triangleq \log \frac{\mathrm{P}\{u_k=0\}}{\mathrm{P}\{u_k=1\}}, \quad \mathrm{P}\{u_k=u\} = \frac{e^{-uL(u_k)}}{1+e^{-L(u_k)}}.$$

The SISO encoder computes the length-N sequence $\mathbf{L}(\mathbf{c}|\mathbf{u}) = (L(c_0|\mathbf{u})\ L(c_1|\mathbf{u})\ \dots\ L(c_{N-1}|\mathbf{u}))^T$ of code bit LLRs $L(c_n|\mathbf{u}) = \log \frac{\mathrm{P}\{c_n=0|\mathbf{L}(\mathbf{u})\}}{\mathrm{P}\{c_n=1|\mathbf{L}(\mathbf{u})\}}$, where $\mathbf{c} = (c_0\ c_1\ \dots\ c_{N-1})^T \in \mathcal{C}$ is the codeword corresponding to the information sequence $\mathbf{u}$. The main task of the SISO encoder is to compute

$$p\big(c_n|\mathbf{L}(\mathbf{u})\big) = \sum_{\sim c_n} p\big(\mathbf{c}|\mathbf{L}(\mathbf{u})\big), \quad n = 0, 1, \dots, N-1, \qquad (1)$$

where $\sim c_n$ implies summation over all code bits except c_n.

The convolutional code $\mathcal{C}$ has a time-invariant trellis representation with M states[1]. In the following, let k denote discrete time and let $S_k \in \{0, 1, \dots, M-1\} = \mathcal{M}$ denote the encoder state at time instant k, where $\mathcal{M}$ denotes the set of states. The encoder has an internal frame structure; the information sequence $\mathbf{u}$ is split into T frames $\mathbf{x}_k = (x_k^{(0)}\ x_k^{(1)}\ \dots\ x_k^{(K_0-1)})^T$ consisting of $K_0 = K/T$ bits each and the codeword $\mathbf{c}$ is similarly split into T frames $\mathbf{y}_k = (y_k^{(0)}\ y_k^{(1)}\ \dots\ y_k^{(N_0-1)})^T$ of length $N_0 = N/T$.

[1] The extension of our discussion to time-varying trellises is straightforward.

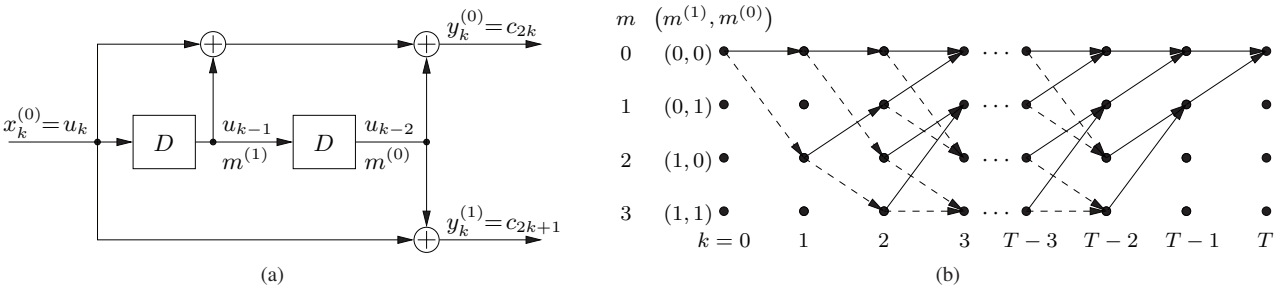

Fig. 2. (a) A rate 1/2 non-recursive convolutional encoder and (b) its trellis diagram. In (b) the solid branches indicate $u_k = 0$ and dashed branches indicate $u_k = 1$.

We now summarize how (1) can be computed using the BCJR algorithm. The BCJR algorithm, adapted to our setting, consists of the following recursions:

$$\alpha_k(m) = \sum_{m'=0}^{M-1} \alpha_{k-1}(m')\gamma_{k-1}(m',m), \tag{2}$$

$$\beta_k(m) = \sum_{m'=0}^{M-1} \beta_{k+1}(m')\gamma_k(m,m'), \tag{3}$$

$$\gamma_k(m',m) = \mathrm{P}\big\{S_{k+1}=m|S_k=m'\big\}. \tag{4}$$

Let m_0 and m_T be the initial state and the terminal state of the encoder. Then the initializations for the forward recursion (2) and the backward recursion (3) are $\alpha_0(m) = \delta_{m,m_0}$ and $\beta_T(m) = \delta_{m,m_T}$, respectively. Here, $\delta_{n,m}$ denotes the Kronecker delta. The state transition probability (4) is in our case given by

$$\gamma_k(m',m) = \prod_{j=0}^{K_0-1} \mathrm{P}\Big\{x_k^{(j)}=b_{m',m}^{(j)}\Big\} = \prod_{j=0}^{K_0-1} \frac{e^{-b_{m',m}^{(j)}L(x_k^{(j)})}}{1+e^{-L(x_k^{(j)})}}, \tag{5}$$

where the transition from state $S_k = m'$ to state $S_{k+1} = m$ happens if and only if $x_k^{(j)} = b_{m',m}^{(j)}$, $j = 0, 1, \ldots, K_0-1$.

With the short-hand notation $p_b\big(y_k^{(j)}\big) = p\big(y_k^{(j)}=b|\mathbf{L}(\mathbf{u})\big)$, the marginalization in (1) can be expressed as

$$p_b\big(y_k^{(j)}\big) = \sum_{(m',m)\in\mathcal{A}_b^{(j)}} \alpha_k(m')\gamma_k(m',m)\beta_{k+1}(m), \tag{6}$$

where $\mathcal{A}_b^{(j)}$ denotes the set of state transitions $S_k = m' \to S_{k+1} = m$ in which the jth bit is equal to b. The above discussion shows that SISO encoding using the BCJR algorithm works essentially in the same way as SISO decoding, except that in (4) we use the prior soft information about the *information* bits rather than of the code bits.

3. EFFICIENT SISO ENCODING

3.1. Forward Recursion Encoder

We next derive a much more efficient implementation of the SISO encoder. Our algorithm is based on the observation that for any time instant and any state there are 2^{K_0} possible outgoing state transitions, yielding a total of $M2^{K_0}$ transitions per time instant, whereas there are only 2^{K_0} different transition probabilities $\gamma_k(m',m)$ at each time instant since according to (5) the transition probabilities depend only on the K_0 information bits. This fact, which is not obvious from the previous equations, will be used below to render the backward recursion (3) superfluous and thereby entail a significant complexity reduction.

Before we discuss the details, we illustrate the idea in terms of a simple example. Fig. 2 shows a rate 1/2 non-recursive convolutional encoder and its trellis diagram. The code is terminated such that $m_0 = m_T = 0$ and $\beta_T(0) = 1$. Due to the termination, the last 2 information bits are zero. Thus the state transition probability up to $k = T-2$ is the same for each branch, i.e., $\beta_{T-2}(m) = 1/4$ for all m. In the following, the backward recursion reads $\beta_k(m) = \beta_{k+1}(m)\sum_{m'}\gamma_k(m,m') = \beta_{k+1}(m)$. Here the important observation is that for all k and for all *reachable* states $m \in \mathcal{M}_k$, we obtain $\beta_k(m) = 1/|\mathcal{M}_k|$, where $\mathcal{M}_k$ denotes the set of states that can be reached at time instant k. It follows that $\beta_k(m)$ is irrelevant for the marginalization in (6). This holds true also for truncated convolutional codes, where $\beta_T(m) = 1/M$ for all $m \in \mathcal{M}$ is the correct initialization of the backward recursion.

We will now simplify the SISO encoding procedure according to the observation from above. Let the posterior state probability at time instant k be denoted by $s_k(m) = \mathrm{P}\{S_k = m|\mathbf{L}(\mathbf{u})\}$ and let $s_0(m) = \delta_{m,m_0}$. The posterior state probabilities of the successor states are then given by

$$s_{k+1}(m) = \sum_{m'\in\mathcal{B}_m} s_k(m')\gamma_k(m',m), \tag{7}$$

where $\mathcal{B}_m$ denotes the set of predecessor states of state $S_{k+1} = m$. Using (7) we can express the posterior probability of the jth bit in the code frame at time k as

$$p_b\big(y_k^{(j)}\big) = \sum_{(m',m)\in\mathcal{A}_b^{(j)}} s_k(m')\gamma_k(m',m). \tag{8}$$

The operation of the efficient *forward recursion encoder* (FRE) can be summarized as follows.

1) Initialize $s_0(m) = \delta_{m,m_0}$, where m_0 is the initial state.
2) As soon as the LLRs $L(x_k^{(j)})$, $j = 0, \ldots, K_0-1$, are available, compute $\gamma_k(m',m)$ using (5) and $L\big(y_k^{(j)}|\mathbf{L}(\mathbf{u})\big) = \log\frac{p_0(y_k^{(j)})}{p_1(y_k^{(j)})}$, $j = 0, \ldots, N_0-1$, using (8); note that $p_1(y_k^{(j)}) = 1 - p_0(y_k^{(j)})$.
3) Compute $s_{k+1}(m)$, $m \in \mathcal{M}$, using (7).
4) Proceed with step 2 until all LLRs are processed.

3.2. Shift-Register Implementation

We next present a very simple and intuitive implementation of (8) in terms of a shift register circuit, which is very well suited for an implementation in hardware. More specifically, we show that a non-recursive convolutional SISO encoder can be implemented using shift registers in the same way as the corresponding HIHO encoder

by replacing the usual modulo-2 sum $\oplus$ with the boxplus operation $\boxplus$ (cf. [8]). The boxplus operation provides a convenient notation to express the reliability of the modulo-2 sum of independent bits. Given two independent bits u_1 and u_2 it follows that

$$L(u_1 \oplus u_2) = L(u_1) \boxplus L(u_2) \triangleq \log \frac{1 + e^{-L(u_1)} e^{-L(u_2)}}{e^{-L(u_1)} + e^{-L(u_2)}}. \quad (9)$$

Clearly, there is $|a \boxplus b| \leq \min\{|a|, |b|\}$, since a combination of bits can never be more reliable than the least reliable individual bit. Since for a recursive encoder the reliability of the (non-systematic) code bit sequence would be monotonically decreasing, we need to restrict the following analysis to non-recursive encoders.

A sketch of the proof for our proposed *shift register encoder* (SRE) is given in the following. For the sake of notational simplicity we restrict ourselves to an encoder with $K_0 = 1$, i.e., $x_k^{(0)} = u_k$. Generalization to the case $K_0 > 1$ is straightforward.

Consider a non-recursive shift register with memory length I and $M = 2^I$ different states. The contents $(m^{(I-1)} \ \dots \ m^{(1)} \ m^{(0)})^T$ of the shift-register serve as binary label for the corresponding state, i.e., $m = \sum_{i=0}^{I-1} m^{(i)} 2^i$. Let the jth generator sequence of the encoder be $\mathbf{g}^{(j)} = (g_0^{(j)} \ g_1^{(j)} \ \dots \ g_I^{(j)})^T$, where $g_i^{(j)} = 1$ if the ith tap is connected to the jth output and $g_i^{(j)} = 0$ otherwise. Note that g_0 corresponds to the connection with the input u_k whereas g_I corresponds to the output of the last delay element. It is not hard to see that the state probability $s_k(m)$ in (7) can be written as

$$s_k(m) = \prod_{i=0}^{I-1} \frac{\exp\big(-m^{(i)} L(u_{k-I+i})\big)}{1 + \exp\big(-L(u_{k-I+i})\big)}. \quad (10)$$

Let us define the set of states for which $y_k^{(j)} = b$ when $u_k = u$, i.e.,

$$\mathcal{M}_{u,b}^{(j)} = \Big\{ m \in \mathcal{M} \,\Big|\, u \oplus \textstyle\sum\!\!\!\!\bigoplus_i m^{(i)} g_{I-i}^{(j)} = b \Big\}. \quad (11)$$

Using (5), (10) and (11) we rewrite (8) as LLR:

$$L\big(y_k^{(j)} | \mathbf{L}(\mathbf{u})\big) = \log \frac{1 + e^{-L(u_k)} e^{-\widetilde{L}_k^{(j)}}}{e^{-L(u_k)} + e^{-\widetilde{L}_k^{(j)}}} = L(u_k) \boxplus \widetilde{L}_k^{(j)},$$

where

$$\widetilde{L}_k^{(j)} = \log \frac{\sum_{m \in \mathcal{M}_{0,0}^{(j)}} s_k(m)}{\sum_{m \in \mathcal{M}_{1,0}^{(j)}} s_k(m)}.$$

The above expression can be shown to equal

$$\widetilde{L}_k^{(j)} = \log \frac{\prod\limits_{i \geq 0 : g_{i+1}^{(j)} = 1} \big[1 + e^{-L(u_{k-I+i})}\big] + \prod\limits_{i \geq 0 : g_{i+1}^{(j)} = 1} \big[1 - e^{-L(u_{k-I+i})}\big]}{\prod\limits_{i \geq 0 : g_{i+1}^{(j)} = 1} \big[1 + e^{-L(u_{k-I+i})}\big] - \prod\limits_{i \geq 0 : g_{i+1}^{(j)} = 1} \big[1 - e^{-L(u_{k-I+i})}\big]},$$

which corresponds to a sequence of boxplus operations on those $L(u_{k-I+i})$ for which $g_{i+1}^{(j)} = 1$. Therefore, we can finally write

$$L\big(y_k^{(j)} | \mathbf{L}(\mathbf{u})\big) = L(u_k) \ \boxplus \!\!\!\!\sum_{i \geq 0 : g_{i+1}^{(j)} = 1}\!\!\!\!\boxplus L(u_{k-I+i}), \quad (12)$$

which is precisely the shift-register implementation with boxplus operations. We note that the restriction to non-recursive encoders is uncritical in distributed channel coding. For example, the BER performance of the relaying scheme presented in [6] does not degrade significantly in case a non-recursive encoder is used at the relay.

4. COMPLEXITY ANALYSIS AND APPROXIMATIONS

We next quantify the computational complexity and the memory requirements for the algorithms discussed in the previous sections. We assume that $\log(\cdot)$ and $\exp(\cdot)$ are evaluated using lookup tables. Thus, the conversion from LLRs to probabilities and vice versa is performed using a table lookup (TL) operation. Multiplications, additions, and table lookups, are counted as one operation each.

4.1. BCJR Encoder

For the BCJR encoder the number of operations can be shown to be given by (underbraces indicate the relevant equations)

$$\mathsf{OPS}_{\mathsf{BCJR}} = \underbrace{2(2M^2 - M)(T-1)}_{(2)+(3)} + \underbrace{2^{K_0} T K_0^+}_{(5)} + \underbrace{(3M 2^{K_0 - 1} - 1) N_0 T}_{(6)} + \underbrace{(2K_0 + N_0) T}_{\text{TLs: } (5)+(6)},$$

where $K_0^+ = \max\{K_0 - 1, 1\}$. The number of states is $M = 2^\nu$, where ν is the constraint length of the code. It is seen that the complexity of the BCJR encoder is linear in the block length and exponential in the constraint length of the code. We note that the BCJR encoding delay is rather large since the output is available only after the forward and the backward recursion have been computed.

The number of memory locations needed by the BCJR encoder is given by $\mathsf{MEM}_{\mathsf{BCJR}} = 2MT + 2^{K_0} T$, where the first term accounts for the forward and the backward recursion and the second term corresponds to the storage of the transition probabilities. We neglect the memory locations needed for the lookup tables and for one section of the time-invariant trellis since this information can be stored in read-only memory. $\mathsf{MEM}_{\mathsf{BCJR}}$ could be reduced down to MT if (5) is recomputed every time it is used and if the output is calculated in reverse order. In any case, the storage size grows linearly with the block length and exponentially with the constraint length.

4.2. Forward Recursion Encoder

For the FRE, the number of operations is given by

$$\mathsf{OPS}_{\mathsf{FRE}} = \underbrace{2^{K_0} T K_0^+}_{(5)} + \underbrace{(2^{K_0+1} - 1)(T-1)}_{(7)} + \underbrace{(M 2^{K_0} - 1) N_0 T}_{(8)} + \underbrace{(2K_0 + N_0) T}_{\text{TLs: } (5)+(8)}.$$

Also for the FRE, the complexity scales linearly with the block length and exponentially with the constraint length. However, observe that the FRE allows the frames to be encoded successively, thereby reducing the encoding delay significantly. Memory is required only for the processing of one frame and hence $\mathsf{MEM}_{\mathsf{FRE}} = 2^{K_0} + 2M$, where the first term corresponds to the state transition probabilities and the second term corresponds to the current and the new state probabilities. The storage size is thus *independent* of the block length and exponential in the constraint length.

4.3. Shift Register Encoder

With the SRE, given by (12), we perform the boxplus operation in the linear (probability) domain, i.e.,

$$\mathrm{P}\{u_0 \oplus u_1 = 0\} = (1 - \mathrm{P}\{u_0 = 0\})(1 - \mathrm{P}\{u_1 = 0\}) + \mathrm{P}\{u_0 = 0\}\,\mathrm{P}\{u_1 = 0\}.$$

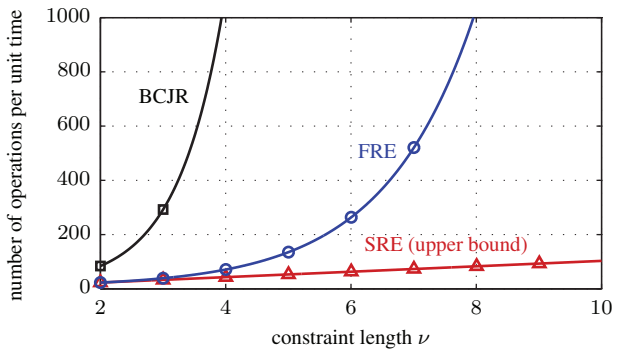

Fig. 3. Number of operations per unit time versus constraint length ν for a rate $1/2$ encoder with $K_0 = 1$ and $N_0 = 2$.

The above expression can be evaluated with 5 operations. The number of boxplus operations depends on the weight of the generator sequences. We can provide a simple upper bound on the number of operations, assuming that the length of each shift register is equal to the memory order I and all taps of all shift registers as well as the inputs are connected to all outputs. Using these assumptions we obtain

$$\mathsf{OPS}_{\mathsf{SRE}} \leq 5N_0[K_0(I+1)-1]T + (K_0+N_0)T, \qquad (13)$$

where the last term in (13) accounts for the TLs necessary to perform the conversions between LLRs and probabilities. Note that the number of operations is *linear in the block length and in the constraint length*. Moreover the required storage is minimal; it equals the constraint length, $\mathsf{MEM}_{\mathsf{SRE}} = \nu$. Hence, the SRE is clearly superior to the other SISO encoders we have discussed in terms of computational complexity and memory requirements.

Fig. 3 depicts a comparison in terms of computational complexity between the BCJR, FRE and SRE algorithms for a rate 1/2 encoder with $K_0 = 1$, $N_0 = 2$ and $\nu = 2, 3, \ldots, 10$. We can see that the FRE is substantially more efficient than the BCJR while the SRE is even more efficient (e.g., at least 12 times more efficient than the FRE at a constraint length of $\nu = 8$).

4.4. Further Simplifications

A further reduction in computational complexity can be achieved in the log-domain by applying suitable approximations. For the FRE we can write

$$s^*_{k+1}(m) = \log \sum_{m' \in \mathcal{B}_m} \exp\big(s^*_k(m') + \gamma^*_k(m', m)\big), \qquad (14)$$

$$p^*_b\big(y^{(j)}_k\big) = \log \sum_{(m',m) \in \mathcal{A}^{(j)}_b} \exp\big(s^*_k(m') + \gamma^*_k(m', m)\big), \qquad (15)$$

where $s^*_k(m) = \log s_k(m)$ and likewise for all other starred quantities. Applying the max-log relation with correction term, given by

$$\log\left(e^a + e^b\right) = \max(a,b) + \log\left(1 + e^{-|a-b|}\right), \qquad (16)$$

to the log-formulation of the FRE, allows for further complexity reduction since the correction term depends only on $|a-b|$ and hence can be evaluated using a small lookup table. A nested version of (16) can be applied onto (14) and (15). By dropping the correction term altogether, a cruder but less complex approximation can be obtained. With (16) we can easily trade off lookup table size against approximation accuracy.

For the SRE we can lower the complexity by using an approximation of the boxplus operator and by working on LLRs directly. The boxplus operation (9) can be expressed as

$$a \boxplus b = a \mathbin{\widetilde{\boxplus}} b \cdot \left(1 - \frac{1}{\min(|a|,|b|)} \log \frac{1 + e^{-\big||a|-|b|\big|}}{1 + e^{-|a|+|b|}}\right), \qquad (17)$$

where $a \mathbin{\widetilde{\boxplus}} b = \operatorname{sign}(a)\operatorname{sign}(b)\min(|a|,|b|)$ and the second term in (17) is a multiplicative correction factor with values between 0 and 1. Note that $a \mathbin{\widetilde{\boxplus}} b \geq a \boxplus b$ overestimates the true result. The correction factor in (17) can be stored in a matrix, yielding a high accuracy approximation with low complexity. Again, the matrix size can be traded off against the quality of the approximation; dropping the correction factor leads to the simplest implementation possible.

5. CONCLUSIONS

We have considered the problem of SISO encoding of convolutional codes. Starting out with the BCJR algorithm, we have shown that its backward recursion is superfluous due to the special structure of the problem. Hence, a forward recursion is sufficient, lowering the complexity of SISO encoding significantly. For the case of non-recursive convolutional encoders, we have proposed an intuitive SISO encoding algorithm based on shift registers, strongly reminiscent of a conventional HIHO encoder. A comparison in terms of computational complexity shows that the SRE outperforms the other encoding methods "hands-down" since its complexity is only linear in constraint length and block length.

REFERENCES

[1] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. 20, pp. 284–287, March 1974.

[3] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE GLOBECOM-89*, Dallas, TX, June 1989, pp. 1680–1686.

[4] J. Hagenauer, "Soft is better than hard," *Communications, Coding and Cryptology*, pp. 155–171, Kluwer Academic Publishers, 1994.

[5] C. Berrou, A. Glavieux, and P. Thitimajshime, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE ICC-1993*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[6] H. Sneessens and L. Vandendorpe, "Soft decode and forward improves cooperative communications," in *1st IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, 2005, pp. 157–160.

[7] P. Weitkemper, D. Wübben, V. Kühn, and K.D. Kammeyer, "Soft information relaying for wireless networks with error-prone source-relay link," in *7th International ITG Conference on Source and Channel Coding*, 2008.

[8] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, March 1996.