A Factor-Graph Schedule for Stream-Oriented Turbo Codes

Alexandre de Andrade and Jaime Portugheis

Abstract— **The paper considers the application of a factor-graph approach to decoding stream-oriented turbo codes. The decoding schedule proposed can be interpreted as only two decoder modules replacing the pipelined structure originally used in stream-oriented decoding. Simulation results corroborate the efficiency of the schedule and also show potential improvements that can be obtained by using MAP instead of Max-Log-MAP calculations.**

Keywords— **Stream-Oriented Turbo Codes, Convolutional Interleaver, Factor-Graphs, Decoding Schedules.**

I. Introduction

Original turbo scheme uses block encoding, where each block of data is encoded and decoded independently. Trellis termination is required in order to obtain the best performances.

In [1]-[5], the stream decoding paradigm was developed. Encoder uses continuous encoding, without explicit block boundaries. In a stream-oriented scheme, encoders are free-running, and its trellises evolve without forcing any state configuration by termination bits or truncation. Stream decoding is proposed to use a pipeline of continuous independent decoding modules, each one using a continuous version of the symbol-by-symbol sliding-window MAP decoding algorithm.

In a sliding window decoding we have information about initial machine states and final states must be learned. This is usually done by performing advanced backward decoding.

In articles where the stream decoding paradigm was developed a more detailed description of decoding calculations is lacking. We focus on the possible optimizations and variations of this calculations through the scheduling analysis of the factor-graph approach.

Recently introduced [6], the factor-graph environment has not been fully explored yet, leaving as open problems several scheduling optimizations. Turbo-like decoding are shown to be a special case of the sum-product algorithm applied to a factor graph that describes the concatenation. Previous articles has translated the traditional turbo decoding sequence of calculations to the corresponding scheduling, which is almost straightforward given the blockwise nature of the graph.

The decoding schedule proposed here can be interpreted as only two decoder modules: one for each constituent encoder. This replaces the pipelined structure originally used in stream-oriented decoding. Although this can be equivalent to the pipelined structure, the scheme of two decoding modules can simplify memory management and allow optimizations more easily.

Alexandre de Andrade e Jaime Portugheis are with Departamento de Comunicações, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Brazil, e-mails: wwx@decom.fee.unicamp.br, jaime@decom.fee.unicamp.br

II. Stream-Oriented Turbo Codes

The stream scheme provides compatibility for turbo codes with any periodic (block or nonblock) interleaver.

The most common type of non-block interleavers is the convolutional interleaver, a very structured non-block interleaver with permutation law

$$\pi_{N,B}(k) = k - BN(k \bmod N),\ k = \ldots, -1, 0, 1, \ldots$$

where N is the period and B is the multiplicity. Concatenating B interleavers, each one with permutation law $\pi_{N,1}(k)$ results in a overall interleaver with permutation law $\pi_{N,B}(k)$.

Generalized convolutional interleaver has a permutation law

$$\pi_f(k) = k - Nf(k)$$

where $f(k)$ is any nonnegative periodic function with period N.

A permutation law $\pi(k)$ is causal if $\pi(k) < k$ for all k.

The interleaver spread, delay or span D for a causal permutation law $\pi(k)$ can be defined as

$$D = \max_k \{k - \pi(k)\}.$$

For a generalized convolutional interleaver we have

$$D = N \max_k f(k)$$

and for a convolutional interleaver

$$D = NB(N-1).$$

As stated in various previous works, the total decoding delay is

$$D_{tot} = I(D + 2W),$$

where W is the length of advanced symbols used in backward calculations, and I is the number of decoding iterations. If a pipelined decoding architecture is assumed, and in addition, processors work with frequencies close to information rates, the decoding delay per iteration should be considered [3], [5].

Since W is small for practical encoders, we have

$$D_{tot} \simeq INB(N-1)$$

for a convolutional interleaver. In this case, D is the decoding delay per iteration.

III. The Proposed Schedule

Figure 1 shows the factor-graph for the serial concatenation of two constituent finite state machines. We prefer to describe the serially concatenated scheme because it has a simpler factor-graph description and it results in a more concise explanation of the algorithm operations. The parallel configuration is similar and can be easily deducted.

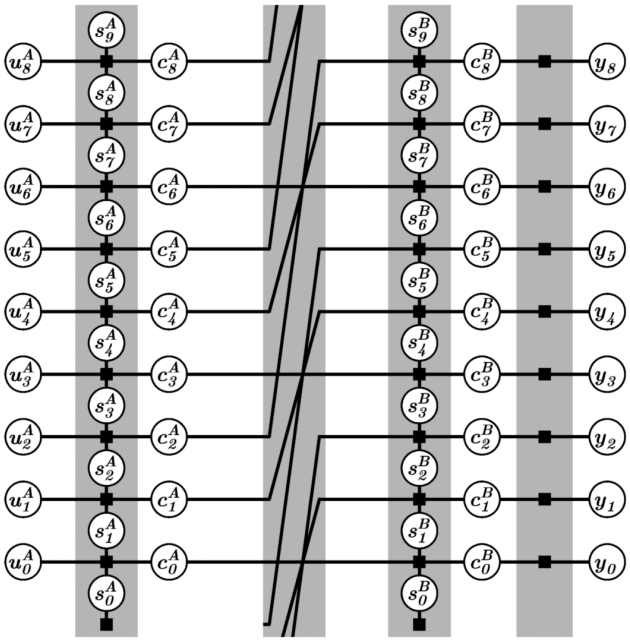

Fig. 1. Factor-graph for the serial concatenation of two codes.

This is actually the initial fragment of the graph, since the whole scheme is semi-infinite, as well as the sequences of variables. The first machine (outer) is defined by the sequence of states $\left\{S_i^A\right\}_{i=0,1,2,\ldots}$ and has as output symbols the sequence $\left\{C_i^A\right\}_{i=1,2,3,\ldots}$. This sequence is interleaved by some permutation law $\pi(i)$ and is passed as input symbols to the second state machine (inner), which has state evolution represented by the sequence $\left\{S_i^B\right\}_{i=0,1,2,\ldots}$. It outputs the symbols $\left\{C_i^B\right\}_{i=1,2,3,\ldots}$ to the channel. The sequence $\{Y_i\}_{i=1,2,3,\ldots}$ is the output of the channel, the received sequence.

The application of the sum-product algorithm to the factor-graph above can be summarized by describing the types of messages on nodes inside state machines. For a general state machine, with input symbols U_i, output symbols C_i and states S_i, we define the following scheme of messages, illustrated in Figure 2.

The sum-product algorithm for this scheme uses as kernel in the central node $T(u_i, s_i, c_i, s_{i+1}) = \Pr\{c_i, s_{i+1} | s_i, u_i\}$, and we have the following update equations for iterative calculations:

$$\alpha(s_{i+1}) = \sum_{u_i, s_i, c_i} T(u_i, s_i, c_i, s_{i+1})\, \alpha(u_i)\, \alpha(s_i)\, \beta(c_i)$$

$$\alpha(c_i) = \sum_{u_i, s_i, s_{i+1}} T(u_i, s_i, c_i, s_{i+1})\, \alpha(u_i)\, \alpha(s_i) \qquad \beta(s_{i+1})$$

$$\beta(s_i) = \sum_{u_i, c_i, s_{i+1}} T(u_i, s_i, c_i, s_{i+1})\, \alpha(u_i) \qquad \beta(c_i)\, \beta(s_{i+1})$$

$$\beta(u_i) = \sum_{s_i, c_i, s_{i+1}} T(u_i, s_i, c_i, s_{i+1}) \qquad \alpha(s_i)\, \beta(c_i)\, \beta(s_{i+1})$$

This set of equations is valid for both constituent decoders, including the directions for the graphical representation of the messages flow. In this unified view, we have two Markov chains $\left(U_i^A, S_i^A, C_i^A\right)$ and $\left(U_i^B, S_i^B, C_i^B\right)$ connected by a interleaver that identifies U_i^B to $C_{\pi(i)}^A$. The interleaver map makes the interface between messages from one decoder to another. Messages $\beta^B(u_i)$ from the decoder B are sent to the decoder A as $\beta^A\left(c_{\pi(i)}\right)$, and in the opposite way messages $\alpha^A(c_i)$ are sent as $\alpha^B\left(u_{\rho(i)}\right)$, where ρ is the mapping function corresponding to π.

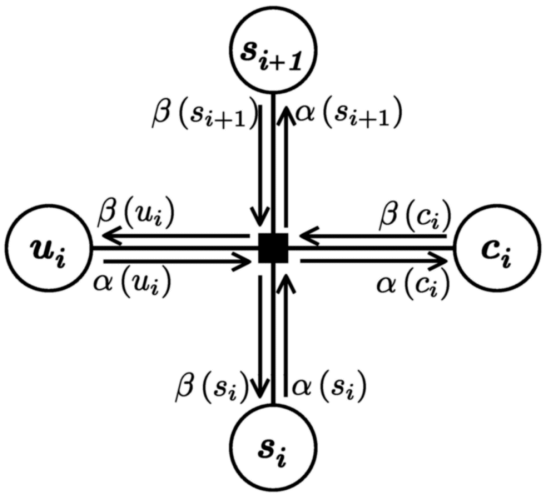

Fig. 2. Illustration of message flow at a function node.

At a given instant k, we have the received sequence $y_0, \ldots y_k$ from the channel. Instead of a symbol by symbol decoding, we chose to decode groups of symbols in a period (the synchronization length).

So, at each N symbols received (one period length), we start calculations that result in N symbols decoded. At receiving channel observations $y_k, \ldots, y_{k+N-1}$ we end up with decisions on the symbols $c_{k-d}^A, \ldots, c_{k+N-1-d}^A$, with $d = D_{tot} = I(D + 2W)$.

The stream-oriented decoding algorithm can be summarized as follows:

(1) receive the channel values corresponding to one period and calculate the a posteriori probabilities $\beta^B(c_i) = p(y_i|c_i)$, $k < i \leq k+N-1$. We use two indexes $k_{\min}$ and $k_{\max}$ to denote the range of calculations inside a period and through iterations. We start with

$$k_{\min} = k$$

and

$$k_{\max} = k + N - 1.$$

The items (a) through (e) describe calculations for the current period:

(a) calculations on *decoder B*

$$\begin{array}{l} \text{evaluate } \alpha^B(s_i), \ \ k_{\min} < i \leq k_{\max} \\ \text{make } k_{\min} = \max\{0, k_{\min} - W\} \\ \text{evaluate } \beta^B(s_i), \ \ k_{\min} < i \leq k_{\max} \\ \text{make } k_{\max} = k_{\max} - W \\ \text{evaluate } \beta^B(u_i), \ \ k_{\min} < i \leq k_{\max} \end{array}$$

(b) here we have the deinterleaver delay:

$$\begin{array}{l} \text{make } k_{\min} = \max\{0, k_{\min} - D\} \\ \text{make } k_{\max} = k_{\max} - D \end{array}$$

(c) calculations on *decoder A*

$$\begin{array}{l} \text{evaluate } \alpha^A(s_i), \ \ k_{\min} < i \leq k_{\max} \\ \text{make } k_{\min} = \max\{0, k_{\min} - W\} \\ \text{evaluate } \beta^A(s_i), \ \ k_{\min} < i \leq k_{\max} \\ \text{make } k_{\max} = k_{\max} - W \\ \text{evaluate } \alpha^A(c_i), \ \ k_{\min} < i \leq k_{\max} \end{array}$$

(d) if we have enough iterations, decide for symbols C_i^A using the product $\alpha^A(c_i)\,\beta^A(c_i)$ in the range $k_{\min} < i \leq k_{\max}$, and restart the process (1) for the next period of symbols; if not enough iterations, go to item (a) (next iteration);

Initial calculations with contour conditions (before stream decoding become stationary) are handled by truncations in $k_{\min}$ to zero (the "max" statements in the description above).

As an elucidative example, we sketch below a simple case with a convolutional interleaver of period $N = 3$, two iterations ($I = 2$) are realized, and the advance window length is $W = 3$. So, we have $D_{tot} = 24$.

In Figure 3, we represent each stage of the decoding procedure as a frame column. Each column is a very simplified visualization of the factor-graph for the serial concatenation, where it is shown just the interleaver connections and the function nodes at the two state machines. Left nodes represent the outer code, and right nodes the inner code. Up arrows are $\alpha(s_i)$ message calculations; down arrows are $\beta(s_i)$ calculations; left directed arrows are $\beta(u_i)$ calculations; and right directed arrows are $\alpha(c_i)$ calculations. Arrows on the left nodes indicate calculations performed on the outer decoder; arrows on the right nodes are calculations on the inner decoder.

IV. Simulation Results

In the previous section we have described a schedule for a serially concatenated turbo code in order to obtain a concise explanation of the algorithm operations. However, in order to compare our performance results with those results of [1], we have decided to implement a similar schedule for a factor-graph representing a parallel concatenation of convolutional codes (PCCCs). All the results are for BPSK signaling over an additive white Gaussian noise channel.

Figure 4 shows the obtained performance for a fixed period $N = 14$ and several values of parameter B. The codes were generated by recursive systematic convolutional (RSC) constituent encoders and puncturing of the parity streams were performed to achieve a code rate $R = 1/2$. Both the encoders and the number of iterations are the same as in Figure 14 of [1]. However, our factor-graph approach corresponds to the use of a sliding-window MAP algorithm at each constituent decoder instead of a sliding-window Max-Log-MAP, as was used in [1]. As can be noticed from the Figure, this yields a better performance of our approach for medium to low error rates. For example, for $E_b/N_0 = 1.5$ dB and $B = 5$, our approach yields an improvement of about two decades in error rate !

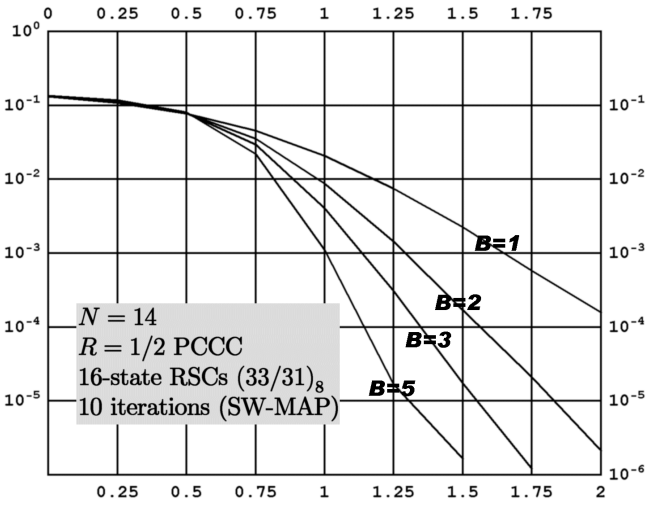

Fig. 4. Bit Error Rate (ordinate) versus E_b/N_0 dB (abscissa) for fixed $N = 14$ and variable B.

Figure 5 shows the obtained performance for a fixed delay $D = 1260$ and several values of parameters N, B, where we have also used the same scheme described in Figure 15 of [1]. Similarly, we have a better performance in comparison to the original algorithm described in [1] (For $E_b/N_0 = 1.5$ dB and $N = 15, B = 6$, our approach yields an improvement of about one decade in error rate).

V. Conclusions

We have developed a factor-graph approach to decode stream-oriented turbo codes. Simulation results were obtained and they corroborate the efficiency of this approach. The sum-product algorithm was applied to the factor-graph in its original form. We expect that a Log-MAP version of

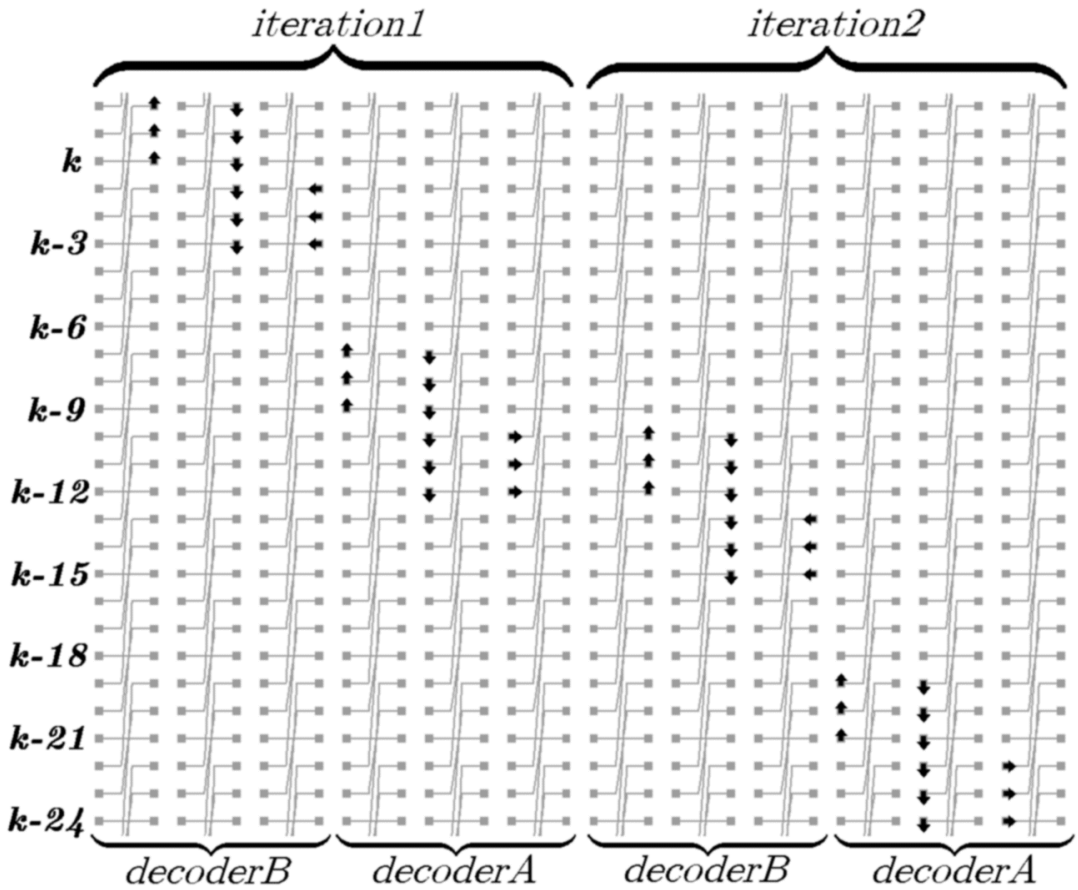

Fig. 3. Graphical representation of the proposed schedule for two iterations at a given instant k.

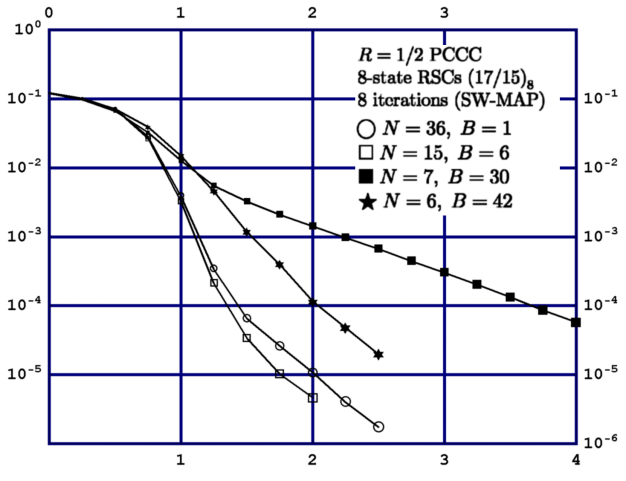

Fig. 5. Bit Error Rate (ordinate) versus E_b/N_0 dB (abscissa) for fixed $D = 1260$ and variable N and B.

the sum-product algorithm (with few values in the correction table) results in negligible performance losses as was the case for the turbo-code system simulated in [7].

The algorithm developed in this paper for the proposed factor-graph schedule uses the concept of a semi-infinite time index for all graph variables. This can be easily implemented by any computer programming language by using the operation of integer modulus on memory addressing. The concept of integer modulus can also lead to a hardware implementation with a circular memory filling. Hence, we believe that the factor-graph representation is more suited in a continuous decoding environment.

References

[1] E. K. Hall and S. G. Wilson, Stream-Oriented Turbo Codes, *IEEE Trans. Inform. Theory*, vol. 47, pp. 1813-1831, July 2001.

[2] E. K. Hall and S. G. Wilson, Stream-Oriented Parallel Concatenated Convolutional Codes, *Proc. IEEE Inform. Theory Workshop*, pp. 4-5, Killarney, Ireland, June 1998.

[3] E. K. Hall and S. G. Wilson, Stream-Oriented Turbo Codes, *Proc. IEEE Vechicular Technology Conf.*, pp. 71-75, Ottawa, Canada, May 1998.

[4] Eric K. Hall and Stephen G. Wilson, Convolutional Interleavers for Stream-Oriented Parallel Concatenated Convolutional Codes, *Proc. IEEE Int. Symp. Inform. Theory*, p. 33, Cambridge, USA, Aug. 1998.

[5] W. Henkel, L. Jusif and J. Sayir, Random Convolutional Interleaving in Turbo Coding, *unpublished work*.

[6] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, Factor Graphs and the Sum-Product Algorithm, *IEEE Trans. Inform. Theory*, vol. 47, pp. 498-519, February 2001.

[7] P. Robertson, E. Villebrun and P. Hoeher, A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in the Log Domain, *Proc. IEEE Int. Conf. on Communications*, pp. 1009-1013, Seattle, USA, June 1995.