Simplified recursion units for Max-Log-MAP: New trade-offs through variants of Local-SOVA

Rami Klaimi, Stefan Weithoffer, Charbel Abdel Nour, Catherine Douillard

Simplified recursion units for Max-Log-MAP: New trade-offs through variants of Local-SOVA

Rami Klaimi, Stefan Weithoffer, Charbel Abdel Nour and Catherine Douillard
IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France
e-mail: firstname.surname@imt-atlantique.fr

***Abstract*—The Log-domain BCJR algorithm is broadly used in iterative decoding processes. However, the serial nature of the recursive state metric calculations is a limiting factor for throughput increase. A possible solution resorts to high-radix decoding, which involves decoding several successive symbols at once. Despite several studies aiming at reducing its complexity, high-radix processing remains the most computationally intensive part of the decoder when targeting very high throughput. In this work, we propose a reformulation specifically targeting the complexity reduction of the recursive calculation units by either limiting the required number of operations or by selectively removing unnecessary ones. We report a complexity reduction of the add-compare-select units in the order of 50% compared to the recently proposed local-SOVA algorithm. In addition, our results show that several performance/complexity trade-offs can be achieved thanks to the proposed simplified variants. This represents a promising step forward in order to implement efficient very high throughput convolutional decoders.**

***Keywords*—Max-Log-MAP decoding, local-SOVA algorithm, low-complexity decoding, high-throughput turbo decoders.**

I. Introduction

The Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [1] or its variants are widely used to compute the maximum a posteriori (MAP) estimate of transmitted symbols in iterative decoders or detectors [2]. Recently, a number of studies have focused on fully parallel or fully pipelined hardware implementations of the Max-Log-MAP (MLM) algorithm [3] to achieve a throughput of tens of Gb/s [4]. The price to pay to reach such high-throughput levels is a significant increase in area due to the computational complexity of MLM decoding [5]. To address this detrimental effect, the local-SOVA algorithm (LSOVA) was introduced in [6]. It relies on a new low-complexity soft-output calculation unit (SOU) that applies a path-based decoding variant of the MLM algorithm. This algorithm was adapted for a hardware implementation in [7] and was shown to significantly reduce the hardware complexity compared to a standard MLM decoder. Therefore, the resulting increased area efficiency enables higher throughput implementations of turbo decoders.

While [6] and [7] focused on the SOU, this work targets the serial nature and complexity of the add-compare-select (ACS) units that calculate the forward and the backward recursions. In this sense, the results presented in the remainder of this paper are complementary to [6], [7]. Increasing the radix order for the ACS units, i.e. processing several trellis sections at the same time, lowers the latency but is associated with an increase in area complexity and a long critical path [8], [9]. Hence, reducing the number of ACS units without increasing the complexity of the performed computations remains an open question. This problem was nevertheless addressed in [10] in the context of non-binary turbo decoders, where a modified bubble sorter was shown to reduce significantly the required number of ACS units. Corresponding results confirmed that the decoder complexity cannot be lowered without reducing the complexity of the recursions.

In this paper, we develop two different LSOVA variants that reduce the complexity of the recursive calculations. While these simplifications do apply on simple trellis-based decoding or detection algorithms, in this work we take turbo decoding as an example to assess the effect of the proposals on the decoding performance. We report significant complexity reductions in comparison with the LSOVA algorithm in [6], [7]. Proposed variants enable further reductions in area complexity and in latency of turbo decoder hardware implementations.

This paper is structured as follows: Section II gives a background overview of the reference decoding algorithms. In Section III, we propose two complementary variants of these algorithms: Section III-A explains how to reduce the complexity of the recursive calculations while Section III-B proposes a technique to compute all the soft decisions in a single step, thus reducing the decoding latency. Performance comparisons in terms of error correction capability and computational complexity are provided in Section IV. Finally, Section V concludes the paper.

II. Reference decoding algorithms

A. The Max-Log-MAP decoding algorithm

In this section, we briefly recall the equations of the MLM decoding algorithm [3]. The forward and the backward metrics α and β for state j at trellis step i are computed using the branch metrics γ as:

$$\alpha_i(j) = \max_{j' \in \mathcal{S}_j} (\alpha_{i-1}(j') + \gamma_{i-1}(j', j)) \tag{1}$$

$$\beta_i(j) = \max_{j' \in \mathcal{S}'_j} (\beta_{i+1}(j') + \gamma_i(j, j')) \tag{2}$$

$\mathcal{S}_j$ and $\mathcal{S}'_j$ are the sets of trellis states at steps $i-1$ and $i+1$, respectively, that are connected to state j at trellis step i.

In addition, the soft output is calculated as a log-likelihood ratio (LLR) at step or time index i :

$$\begin{aligned} \mathrm{L}_i = &\max_{(j,j') \in \{0 \cdots 2^\nu\}^2 \,|\, s(j,j')=0} (\alpha_i(j) + \beta_{i+1}(j') + \gamma_i(j, j')) \\ &- \max_{(j,j') \in \{0 \cdots 2^\nu\}^2 \,|\, s(j,j')=1} (\alpha_i(j) + \beta_{i+1}(j') + \gamma_i(j, j')) \end{aligned} \tag{3}$$

where ν is the memory of the considered convolutional code and $s(j, j')$ is the systematic bit that connects state j at trellis step i to state j' at trellis step $i+1$, if the connection exists. When aiming for high throughput, several trellis sections can be decoded together. Precisely, a radix-2^q trellis allows the decoding of q bits at each decoding step. Equations (1), (2) and (3) have then to be adapted accordingly. Additionally, for high-throughput decoding, the data frames are in practice decomposed into windows of size W, processed in parallel.

B. The local-SOVA decoding algorithm

The LSOVA is a path-based variant of the MLM algorithm introduced in [6]. It uses the same recursive metrics calculations (1) and (2) but the LLRs are computed using path-based local update rules inspired by the SOVA. Conventionally, a path in a trellis diagram is defined as a sequence of states and is associated with an input bit sequence and a path metric. In [6], an alternative definition of a path, with a more local sense that focuses on a particular trellis section is adopted. When considering a radix-2^q trellis, for each trellis section, a path is defined for each state j as a 3-tuple consisting of the path metric M, the q hard decisions $u_j = \{u_{j,0}, \ldots, u_{j,q-1}\}$ labeling this path, and their associated reliability values $L_j = \{L_{j,0}, \ldots, L_{j,q-1}\}$. Two paths P_a and P_b can be merged into path P_c according to the following:

$$M_c = f_0(M_a, M_b) = \max(M_a, M_b) \tag{4}$$

$$u_c(l) = f_1(u_a(l), u_b(l)) \forall l \in [0, q-1] \tag{5}$$

where f_1 selects the hard decision of the winning path resulting from (4). The reliabilities L are updated following:

$$L_c(l) = f_2(L_a(l), L_b(l)) \forall l \in [0, q-1] \tag{6}$$

with f_2 being the Hagenauer rule (HR) [11] or the Battail rule (BR) [12] depending on $u_a(l) = u_b(l)$ or $u_a(l) \neq u_b(l)$. Let $p = \arg\max(M_a, M_b)$, $p' = \arg\min(M_a, M_b)$ and $\Delta = L_p(l) - L_{p'}(l)$:

$$L_c(l) = \begin{cases} \min(L_p(l), \Delta + L_{p'}(l)) & \text{if } u_a(l) = u_b(l) \text{ (BR)} \\ \min(L_p(l), \Delta) & \text{if } u_a(l) \neq u_b(l) \text{ (HR)} \end{cases} \tag{7}$$

A merge and update tree is traversed to get the final hard and soft outputs.

III. Lowering the complexity of the forward and backward recursions

Forward and backward recursions are performed serially for the W trellis stages within a window. Corresponding computations require a proportional amount of ACS units, depending on the radix order q, the number of states 2^ν and the targeted frame size K. For large W and K, this number becomes prohibitive for fully pipelined/parallel hardware architectures, motivating the study of alternative algorithms with a reduced number and/or simplified ACS units. To address this problem, we propose two low-complexity ACS computation variants.

Fig. 1 shows the conventional computation of the forward and backward recursions launched simultaneously from the edges of a size-W window in a radix-4 trellis. The straight

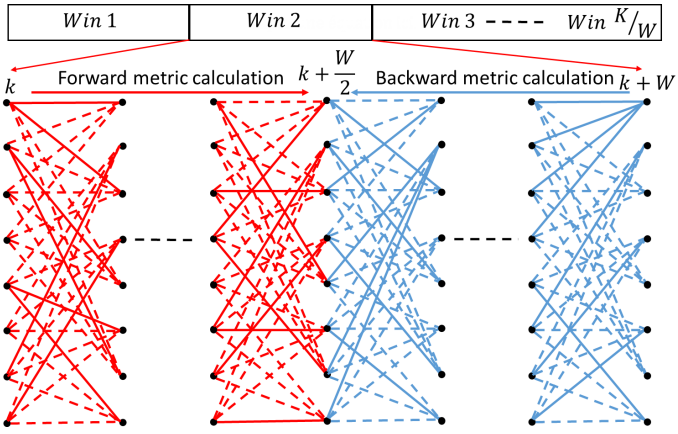

Fig. 1: Windowed MLM decoding in a radix-4 trellis ($q = 2$).

lines correspond to the winning transitions in (1) and (2), while the dotted lines correspond to the losing transitions. Once both recursions meet in the middle (MIM) of the window, i. e. when $\alpha_{k+W/2}$ and $\beta_{k+W/2}$ are computed, two possible strategies can be applied, corresponding to the two proposed simplified variants. The first strategy applies low-complexity recursion units beyond $k + W/2$ to return to the edges of W (down to k for β and up to $k + W$ for α). This strategy is denoted by low complexity return (LCR). The second strategy halts the execution of the recursion units and applies LSOVA over a unique merge tree to update the soft output (reliability) values for the $W/2$ trellis steps of the forward and backward recursions jointly. Indeed, the decoding decision of MLM at $k + W/2$ is the same regardless from the path direction (forwards from k to $k+W/2$ or backwards from $k+W$ to $k+W/2$). This second strategy is called modified dual-sided (MDS) LSOVA. In the remainder of this paper, since the processing is limited to a window, index k is omitted ($k = 1$) for simpler notations.

A. Meet in the middle and low-complexity return

Following Fig. 2, the metric difference between transition l and the winning transition converging to state j at trellis section i is denoted by $\delta_i^l(j)$ for the forward recursion, whereas the metric difference between this same transition and the one that diverges from state j is denoted by $\Delta_i^l(j)$ for the backward recursion. For each trellis stage i while performing

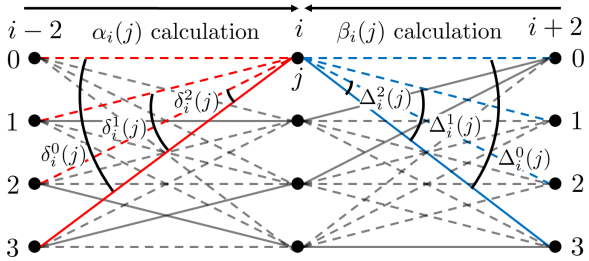

Fig. 2: $\delta_i^l(j)$ and $\Delta_i^l(j)$ for a radix-4 trellis. Straight and dotted lines correspond to winning and losing transitions.

the recursive calculations until MIM, we propose to memorize $\delta_i^l(j)$ and $\Delta_i^l(j)$ instead of the intermediate result of the forward and the backward recursions. Beyond MIM, in contrast

to the MLM algorithm where values of $\alpha_i(j)$ and $\beta_i(j)$ are computed accurately and separately for the trellis stages, an estimate of $\tau_i(j) = \alpha_i(j) + \beta_i(j)$ is computed recursively in our LCR proposal. By considering solely winning transitions from the forward and backward recursions before MIM, this proposal largely reduces computational complexity.

Low-complexity backward τ_i computation beyond MIM: Continuing with the same example, let us consider the calculation of $\tau_i(j), j \in [0,3]$ from the four $\tau_{i+2}(j'), j' \in [0,3]$ values beyond MIM. The corresponding four transitions per state were already considered, with one labeled as winning in the forward metric calculation of α_{i+2}, before MIM. This is illustrated in the example of Fig. 3. Now for the same transitions, considering the opposite direction from the viewpoint of the states at trellis step i, three labelling configurations can be observed: denoting by $n_i(j)$ the number of winning transitions emerging from state j at position i, we have $n_i(0) = 0$, $n_i(1) = n_i(3) = 1$ and $n_i(2) = 2$.

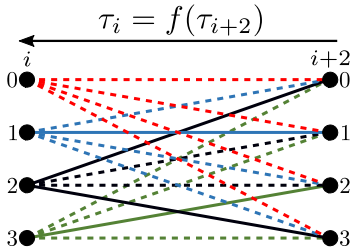

Fig. 3: Example of winning (bold) and loosing (dotted) transitions for the backward computation of τ_i beyond MIM.

When $n_i(j) > 0$, $\tau_i(j)$ can be directly deduced exclusively from τ_{i+2}. When $n_i(j) = 0$, a readjustment calling upon memorized $\delta_i^l(j)$ and $\Delta_i^l(j)$ values, representing the difference between the metrics of currently considered and the winning transitions, is needed to estimate $\tau_i(j)$:

$$\tau_i(j) = \begin{cases} \tau_{i+2}(l_1) & \text{if } n_i(j) = 1 \\ \max(\tau_{i+2}(l_1)) & \text{if } n_i(j) > 1 \\ \max(\tau_{i+2}(l_2)) - \delta_i^j(l_2) & \text{if } n_i(j) = 0 \end{cases} \quad (8)$$

where l_1 is the set of winning transitions emerging from state j during the calculation of α_{i+2} and l_2 is the set of all the transitions emerging from state j.

Low-complexity forward τ_i computation beyond MIM: Similarly, τ_i at the right side of the MIM calls upon transitions labeled as winning in the backward metric calculation of β_{i-2}:

$$\tau_i(j) = \begin{cases} \tau_{i-2}(l_1) & \text{if } n_i(j) = 1 \\ \max(\tau_{i-2}(l_1)) & \text{if } n_i(j) > 1 \\ \max(\tau_{i-2}(l_2)) - \Delta_i^j(l_2) & \text{if } n_i(j) = 0 \end{cases} \quad (9)$$

Using (8) and (9), τ_i is estimated at each position in the sub-trellis and is used in the LSOVA SOU to compute the soft output values. With this proposal, we slightly increase the memory requirements of the decoder, while reducing the performed operations after MIM: the number of comparisons needed for the metric computations is greatly decreased. Moreover, no branch metric memorization/re-computation is required. These reductions are expected to lead to significant improvements of both the complexity and efficiency of resulting hardware architectures. To sum up, the proposed reformulation of the MLM recursive units noticeably reduces the computational complexity while still suffering from the serial nature of these computations.

B. Meet in the middle and modified dual-sided LSOVA

Contrary to the LCR approach, the proposed MDS-LSOVA algorithm avoids the computation of the recursive metrics after MIM, without significant impact on the error correcting performance. To do so, we compute $\alpha_i(j)$ and $\beta_i(j)$ using paths $P_j^b = \{M_j, u_j^b, L_j^b\}$ and $P_j^f = \{M_j, u_j^f, L_j^f\}$ resulting from the backward and the forward recursion computations for each state j at time index i before MIM. A first approach to carry out these computations in one shot is by considering a radix-$2^{W/2}$ LSOVA for the computation of $\alpha_{k+W/2}$ and $\beta_{k+W/2}$. Hence, no path merge occurs before MIM. Then, only one LSOVA SOU merge tree is required to update both sides of MIM while avoiding recursive computations after MIM.

However, the computational complexity of a recursive metric α or β increases exponentially with the number of trellis sections q within a radix. With $q = W/2$, the complexity becomes prohibitive for a practical implementation with typical values of W (32 or 64). As an alternative, we propose to use a lower radix order (typically 4, i. e. $q = 2$) for the recursions before MIM, while still applying only one SOU at MIM. This requires modifications to the LSOVA algorithm due to the merges that occur after each lower-order radix during the recursions in this case. Indeed, if we consider for example a forward recursion with radix 4, it happens that a state at position i does not lead to any winning transition for the merge at $i+2$ (i. e. $n_i(j) = 0$) as seen in Fig. 3. Then, the forward metric accumulated (before i) up to this state will fail to reach the merge tree of the SOU at MIM and will not be considered for the computation of the final reliability values, entailing large performance penalties. To address this issue, we propose to update the reliability values L for all previous trellis steps during the computation of the recursion metrics for each new radix-2^q trellis segment. In the case of forward recursions, let $P_1^f(i)$, $P_2^f(i)$, ... $P_{2^q}^f(i)$ be the paths at trellis position i to be merged to compute $P_{out}^f(i+q)$ at trellis position $i+q$. Then, $\alpha_{i+q} = M_{out}^f$ is computed from α_i, and $i+q$ bits are to be updated according to (6). The merge operation computes

$$M_{out}^f = f_0(M_1, \ldots, M_{2^q}) = \max(M_1^f, \ldots, M_{2^q}^f) \quad (10)$$

$$u_{out}^f(l) = f_1(u_1^f(l), \ldots, u_{2^q}^f(l)) \forall l \in [1, i+q]. \quad (11)$$

The reliability values are updated through

$$L_{out}^f(l) = f_2(L_1^f(l), \ldots, L_{2^q}^f(l)) \forall l \in [1, i+q] \quad (12)$$

where f_1 and f_2 are defined as in (5) and (6).

Similar merge operations are to be performed during the backward recursions. At MIM, the 2^ν length-$W/2$ winning paths of the forward and backward recursions meet. The result

of the addition of their metrics M^f and M^b is provided as input to a LSOVA SOU tree involving $2^\nu - 1$ merge operations, generally organized in ν layers [6]. Each merge operation involves updating the reliability values of the hard decisions labeling the merged paths on both sides of MIM, due to symmetry between the forward and backward processes.

The number of performed updates has a great impact on computational complexity. Aiming to reduce it, we studied the effect of limiting the update depth. This was motivated by the well-known feature of convolutional codes that all trellis paths merge to the maximum likelihood path after some trellis steps. When considering no limitation, the number of reliability updates to be performed for each trellis state when processing trellis position i in the forward recursion, $n^f_{upd}(i)$, is i. Therefore, when considering radix-2^q processing for the recursions, the total number of updates performed during the forward recursions is

$$N^f_{upd} = 2^\nu \sum_{m=1}^{\frac{W}{2^q}} n^f_{upd}(mq) = 2^\nu \frac{q}{2}\frac{W}{2^q}\left(\frac{W}{2^q}+1\right) \qquad (13)$$

Due to symmetry, the total number of updates performed during the backward recursions N^b_{upd} is also equal to N^f_{upd}. In addition, the total number of updates in the SOU is

$$N^{SOU}_{upd} = (2^\nu - 1)W \qquad (14)$$

In summary, MDS-LSOVA replaces half of the recursive computations by updates and can thus largely limit latency since the necessary update operations can be highly parallelized and pipelined. Moreover, limiting the number of updates can have a significant impact on complexity and error correction performance. Several trade-offs were assessed when limiting the value of n^f_{upd} to different fixed values in (13) as well as the number of updates in (14), depending on the layer s, $s \in [1, \nu]$. They are illustrated in the following example.

IV. Case study: simulation results and complexity analysis

In this section, we consider a turbo code using tail-biting $\nu = 3$ recursive systematic convolutional code with generator polynomials $(1, 15/13)_8$ as component code. A frame size $K = 128$ bits, window size $W = 32$ and an almost regular permutation designed following [13] are used. Table I shows the different configurations for the truncation of the updates. $n^{SOU}_{upd}(s)$, $s \in [1, 3]$ is the number of updates performed by the SOU for each merge operation in layer s (instead of W).

TABLE I: Proposed MDS-LSOVA update truncation configurations.

Conf. \ Upd.	n^f_{upd}	$n^{SOU}_{upd}(1)$	$n^{SOU}_{upd}(2)$	$n^{SOU}_{upd}(3)$
C_1	4	4	4	4
C_2	8	8	8	8
C_3	4	4	4	16
C_4	8	8	8	16
C_5	4	8	8	16
C_6	4	16	16	16
C_7	8	16	16	16

As explained above, forward and backward recursions stop at MIM for MDS-LSOVA. Therefore, the starting and ending positions of the decoding windows have to be shifted between the successive iterations of the decoding process. This can be easily done with a tail-biting code. The starting and ending positions used for each window in our simulation setup are shown in Table II. For initialization, we used the metric values at MIM as initial values for the next iteration, as in the next iteration initialization technique [14].

TABLE II: Window boundaries for odd/even iter. ($K = 128, W = 32$).

Iteration \ Win	Win1	Win2	Win3	Win4
odd	[0, 31]	[32, 63]	[64, 95]	[96, 127]
even	[16, 47]	[48, 79]	[80, 111]	[112, 15]

Fig. 4 shows the error correction performance in terms of bit error rate (BER) of the proposed algorithms in AWGN channel using BPSK modulation. Two coding rates, $R = 1/3$ and $R = 8/9$, were simulated. For both rates, the LCR algorithm and the full-complexity MDS-LSOVA perform within 0.2 dB of the reference LSOVA, which is strictly equivalent to the MLM algorithm. Observed penalties are due to the simplified recursion metrics in the case of LCR and to the limited propagation of the forward and backward metrics in a window (only up to $W/2$ instead of W) during one iteration for the full-complexity MDS-LSOVA. Regarding the configurations with a limited update length, performance is gradually degrading from C_7 to C_1, up to 1 dB compared to full-complexity MDS-LSOVA performance. To choose between configurations, complexity must be taken into account.

The complexity of the different algorithms whose BER performance is shown in Fig. 4 is provided in Table III. It is expressed in number of ACS units and updates in the case of radix-4 processing ($q = 2$). Before MIM, the LSOVA and LCR perform $2^\nu(2^q - 1)\frac{W}{2q} = 192$ ACS operations and updates for each recursion, forward or backward. For the MDS-LSOVA, the same amount of ACS operations is required but the updates are carried out on a certain amount of previous trellis steps, depending on the truncation length. After MIM, there are no more update operations in the recursions and the number of ACS computations is the same as before MIM for the reference LSOVA, while it is statistically halved for the LCR algorithm according to (8) and (9). In the LSOVA and LCR algorithms, $\frac{W}{q} = 16$ SOUs are required, each SOU performing $2^\nu - 1 = 7$ ACS operations, each comparison being followed by $q = 2$ updates (one for each decoded bit). The MDS-LSOVA no longer needs any recursive computation after MIM and requires only one SOU for the whole window, involving $2^\nu - 1 = 7$ ACS operations and $(2^\nu - 1)W = 224$ updates for the full-complexity algorithm. The complexity of the simplified variants of the MDS-LSOVA is computed according to the parameters given in Table I.

All the proposed algorithms reduce the number of ACS units compared to the reference LSOVA. The LCR and MDS-LSOVA algorithms reduce the number of ACS operations by more than 20% and 50%, respectively, at the price of an increased number of updates in some cases. All the simplified MDS-LSOVA configurations require a lower total

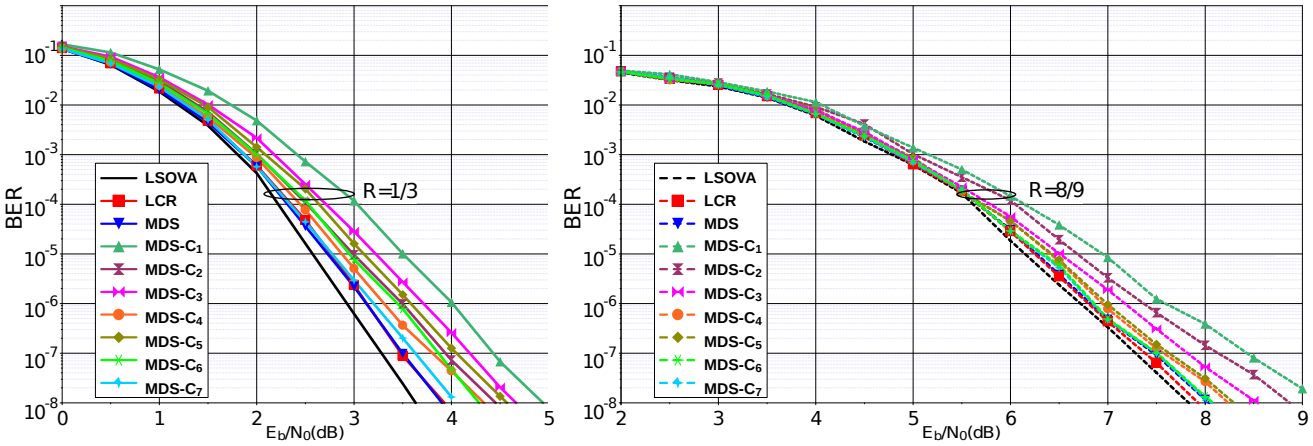

Fig. 4: BER comparison of a turbo code using proposed and reference decoding algorithms. $K = 128$ bits, $W = 32$, code rates $R = 1/3$, $R = 8/9$ and 8 iterations. AWGN channel and BPSK modulation.

TABLE III: Complexity comparison of the considered decoding algorithms in number of ACS and update operations required to decode one window. Radix 4 ($q = 2$), $W = 32$.

Algo \ Unit	Recursion		SOU		Compl./LSOVA		
	ACS	Upd.	ACS	Upd.	ACS	Upd.	Tot. Ops.
LSOVA	768	384	112	224	-	-	-
LCR	576	384	112	224	-21.8%	0%	-12.91%
MDS	384	1152	7	224	-55.6%	+55.8%	+15.8%
MDS-C_1	384	480	7	56	-55.6%	-11.8%	-37.7%
MDS-C_2	384	832	7	112	-55.6%	+35.6%	-11.3%
MDS-C_3	384	480	7	80	-55.6%	-7.9%	-36.1%
MDS-C_4	384	832	7	128	-55.6%	+36.7%	-9.2%
MDS-C_5	384	480	7	128	-55.6%	0%	-32.82%
MDS-C_6	384	480	7	224	-55.6%	+13.6%	-26.5%
MDS-C_7	384	832	7	224	-55.6%	+42.4%	-2.8%

number of operations than LSOVA. Therefore, the choice of the appropriate configuration is a compromise between error correction performance and complexity/latency reduction. For example, the LCR decoder offers a significant reduction in number of ACS units while maintaining the same latency as the LSOVA decoder. Differently, MDS-C_7 performs close to the reference algorithm, with an expected halved latency but with only a slight reduction in complexity.

V. Conclusion

We proposed several variants of the LSOVA algorithm with several performance/complexity/latency trade-offs. The LCR algorithm reduces the complexity of the ACS units by more than 20% for a penalty of up to 0.2 dB depending on the code rate. The second variant goes further to halve the number of ACS units at the cost of more update operations for the soft output computation in certain cases. The decoder designer is offered then a much wider panel of trade-offs compared to prior art. In fact, recent works on LSOVA showed that path-based complexity reductions directly translate to reductions in area complexity for hardware implementation [7]. The results of this paper confirm that path-based variants of the BCJR algorithm have a high potential in achieving low latency decoding, while guaranteeing high throughput.

Acknowledgment

This work was partially funded by the French National Research Agency TurboLEAP project (ANR-20-CE25-0007).

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. Inform. Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE Int. Conf. Commun. (ICC)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[3] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE Int. Conf. Commun. (ICC)*, vol. 2, June 1995, pp. 1009–1013.

[4] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2762–2775, 2015.

[5] S. Weithoffer, R. Klaimi, C. Abdel Nour, N. Wehn, and C. Douillard, "Fully pipelined iteration unrolled decoders-The road to Tb/s turbo decoding," in *IEEE Int. Conf. Acoustics, Speech, and Signal Proc. (ICASSP)*, Barcelona, Spain, May 2020.

[6] V. H. S. Le, C. Abdel Nour, E. Boutillon, and C. Douillard, "Revisiting the Max-Log-Map algorithm with SOVA update rules: new simplifications for high-radix SISO decoders," *IEEE Trans. Commun.*, vol. 68, no. 4, pp. 1991–2004, 2020.

[7] S. Weithoffer, R. Klaimi, C. Abdel Nour, N. Wehn, and C. Douillard, "Low-complexity computational units for the local-SOVA decoding algorithm," in *IEEE 31st Int. Symp. Personal, Indoor and Mobile Radio Commun. (PIMRC)*, London, UK, Sept. 2020.

[8] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *IEEE Custom Integrated Circ. Conf.*, San Jose, CA, USA, 2009, pp. 487–490.

[9] O. Sánchez, C. Jégo, M. Jézéquel, and Y. Saouter, "High speed low complexity radix-16 Max-Log-MAP SISO decoder," in *IEEE Int. Conf. Electron., Circ., and Sys. (ICECS)*, Seville, Spain, 2012, pp. 400–403.

[10] R. Klaimi, C. Abdel Nour, C. Douillard, and J. Farah, "Low-complexity decoders for non-binary turbo codes," in *10th Int. Symp. on Turbo Codes Iter. Inf. Proc. (ISTC)*, Hong Kong, China, Dec. 2018.

[11] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *1989 IEEE Global Telecomm. Conf. and Exhib. (Globecom)*, Nov 1989, pp. 1680–1686 vol.3.

[12] G. Battail, "Pondération des symboles décodés par l'algorithme de Viterbi," in *Annal. Telecomm.*, vol. 42, no. 1-2, 1987, pp. 31–38.

[13] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard, "Protograph-based interleavers for punctured turbo codes," *IEEE Trans. Commun.*, vol. 66, no. 5, pp. 1833–1844, 2018.

[14] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circ. Conf. (ISSCC)*, San Francisco, CA, USA, 2003, pp. 150–484.