Revisiting the Max-Log-Map algorithm with SOVA updates rules: new simplifications for high-radix SISO decoders

Vinh Hoang Son Le, Charbel Abdel Nour, Emmanuel Boutillon, Catherine Douillard

Revisiting the Max-Log-Map algorithm with SOVA updates rules: new simplifications for high-radix SISO decoders

Vinh Hoang Son Le, *Student Member, IEEE,* Charbel Abdel Nour, *Member, IEEE,*
Emmanuel Boutillon, *Senior Member, IEEE* and Catherine Douillard, *Senior Member, IEEE*

***Abstract*—This paper proposes a new soft-input soft-output decoding algorithm particularly suited for low-complexity high-radix turbo decoding, called local soft-output Viterbi algorithm (local SOVA). The local SOVA uses the forward and backward state metric recursions just as the conventional Max-Log MAP (MLM) algorithm does, and produces soft outputs using the SOVA update rules. The proposed local SOVA exhibits a lower computational complexity than the MLM algorithm when employed for high-radix decoding in order to increase throughput, while having the same error correction performance even when used in a turbo decoding process. Furthermore, with some simplifications, it offers various trade-offs between error correction performance and computational complexity. Actually, employing the local SOVA algorithm for radix-8 decoding of the LTE turbo code reduces the complexity by 33% without any performance degradation and by 36% with a slight penalty of only 0.05 dB. Moreover, the local SOVA algorithm opens the door for the practical implementation of turbo decoders for radix-16 and higher.**

***Index Terms*—convolutional codes, soft-input soft-ouput decoding, soft-ouput Viterbi algorithm, high-radix decoding, turbo codes, high throughput.**

I. Introduction

TURBO codes were first proposed by Berrou *et al.* in 1993 [1]. A turbo encoder consists of two binary recursive systematic convolutional (RSC) encoders separated by an interleaver. At the receiver, each component decoder uses a soft-input soft-output (SISO) decoding algorithm to compute extrinsic estimates for every systematic bit. Extrinsic information is exchanged between the component decoders through an iterative process, until convergence is achieved. The Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [2] is usually employed to compute the maximum a posteriori (MAP) estimate of the bits. However, in practice, applying the BCJR algorithm in the logarithmic domain (Log-MAP algorithm) and possibly simplifying it using the max-log approximation (Max-Log-MAP or MLM algorithm), makes it more suitable for hardware implementations [3].

For the past two decades, binary turbo codes have been adopted as forward error correcting (FEC) codes in several wireless communication standards, such as the third and fourth generations (3G and 4G) of wireless mobile telecommunications and WiMAX, but also in digital video broadcasting standards such as DVB-RCS/RCS2 and DVB-SH [4]. In the near future, the evolutions of LTE Advanced Pro will require turbo codes able to achieve throughputs as high as several tens of Gbit/s, so as to complement the 5G New Radio deployments. To this end, the original BCJR algorithm and its derivatives are penalized by their inherent serial nature, thus leading to high latency and low throughput while requiring a large amount of memory to store the state metrics. Therefore, a number of techniques have been proposed as potential solutions to these issues, such as the sliding window technique [5], shuffled parallelism [6], sub-block parallelism [7], full parallelism [8] or high-order radix decoding [9]. In this work, we mainly focus on how to enable high-order radix decoding schemes for turbo decoders. Previous works in the literature, such as [10], [11], mainly tried to increase the throughput of BCJR-based SISO decoders, without specifically considering the complexity reduction of the studied algorithm. Only in [12], a low-complexity radix-16 SISO decoder for the MLM algorithm was proposed, with the introduction of specific processing to limit the resulting error correction degradation at high signal-to-noise ratios.

On the other hand, the soft-output Viterbi algorithm (SOVA) [13] has been recently reconsidered as an efficient SISO candidate for turbo decoding [14], [15]. However, the interest in this algorithm remains limited. This is because, first of all, the MLM algorithm outperforms the SOVA by about 0.75 dB when used for turbo decoding [3] and, second, the serial nature of the SOVA is even more pronounced when compared to the MLM algorithm due to the involved traceback and the soft output update procedures. Nevertheless, the SOVA provides an alternative way to perform the soft output computation for SISO decoding. More specifically, once the forward and backward state metric recursions have been processed, two update rules employed in SOVA decoding, namely the Hagenauer rule [13] and the Battail rule [16], [17], can also be employed to produce the same soft output estimate as the MLM algorithm. This paper shows that using SOVA update rules allows us to perform high-radix decoding of convolutional codes in an efficient way, enabling implementations of high-throughput turbo decoders.

The rest of this paper is organized as follows. Section II recalls the MLM algorithm and analyzes it from the SOVA perspective. Section III describes the local SOVA and shows that the MLM algorithm is an instance of the proposed algorithm. High-radix turbo decoders using local SOVA are then considered with new simplifications in Section IV and simulation results and computational complexity analysis are provided to illustrate the advantages of the local SOVA.

Finally, Section V concludes the paper.

II. Revisiting the MLM Algorithm

Let us assume a binary convolutional encoder with rate $1/m$ and memory length ν. Considering an input message sequence of length K, $\mathbf{u} = (u_0, \ldots, u_{K-1}) \in \{0,1\}^K$, the encoder produces codeword $\mathbf{x} = (x_0, \ldots, u_{M-1}) \in \{0,1\}^M$, where $M = m \times K$. A common representation of a convolutional code is the trellis diagram. With K message bits, the trellis diagram consists of $K+1$ *time indexes*, ranging from 0 to K. At time index k, the state of the convolutional code can take 2^ν values. The interval between a time index k and $k+1$ is considered as *trellis section* k. In a trellis section k, there are branches connecting states at time index k to states at time index $k+1$. Since the considered convolutional code is binary, two branches come out of any state s at time index k, each being connected to a state at time index $k+1$. Of the two branches, one is associated with input bit $u_k = 0$ and the other with input bit $u_k = 1$.

Assuming a transmission using BPSK modulation, the received sequence is $\mathbf{y} = \{y_1, \ldots, y_M\}$. The BCJR algorithm and its derivatives estimate the log-likelihood ratio (LLR) for each systematic bit as [3]:

$$L_{\text{BCJR}}(u_k) = \ln \frac{\sum_{s,s'|u_k=1} Pr\{s_k = s, s_{k+1} = s', \mathbf{y}\}}{\sum_{s,s'|u_k=0} Pr\{s_k = s, s_{k+1} = s', \mathbf{y}\}}. \quad (1)$$

Based on the max-log approximation, $\ln(\sum_{i=1}^{N} x_i) \approx \max_{i=1,\ldots,N}\{\ln(x_i)\}$, (1) can be rewritten so as to derive the MLM LLR estimate, which is the soft output of the decoder [3]:

$$\begin{aligned} L_{\text{MLM}}(u_k) &= \max_{(s,s')|u_k=1} \ln Pr\{s_k = s, s_{k+1} = s'|\mathbf{y}\} \\ &\quad - \max_{(s,s')|u_k=0} \ln Pr\{s_k = s, s_{k+1} = s'|\mathbf{y}\} \\ &= \max_{(s,s')|u_k=1} \big(A_k(s) + \Gamma_k(s,s') + B_{k+1}(s')\big) \\ &\quad - \max_{(s,s')|u_k=0} \big(A_k(s) + \Gamma_k(s,s') + B_{k+1}(s')\big), \end{aligned} \quad (2)$$

where $\Gamma_k(s,s')$ is the branch metric from state $s_k = s$ to state $s_{k+1} = s'$ at trellis section k, $A_k(s)$ is the forward state metric for $s_k = s$ at time index k and $B_{k+1}(s')$ is the backward state metric for $s_{k+1} = s'$ at time index $k+1$.

The forward (respectively backward) state metrics from time index 0 to K (respectively K to 0) are recursively calculated using (3a) (respectively (3b)):

$$A_{k+1}(s) = \max_{s'} \big\{A_k(s') + \Gamma_k(s',s)\big\}, \quad (3a)$$

$$B_k(s) = \max_{s'} \big\{B_{k+1}(s') + \Gamma_k(s,s')\big\}. \quad (3b)$$

The procedure of recursively calculating the forward state metrics is referred to as *forward propagation* and the procedure of recursively calculating the backward state metrics as *backward propagation*.

Let us take as an example the estimation of the soft output related to u_k at trellis section k in Fig. 1, using the MLM algorithm. The forward and backward propagations provide all the values for $A_k(s)$, $s = 0, \ldots, 3$ and $B_{k+1}(s')$, $s' = 0, \ldots, 3$. Then, assuming that $(s,s') = (0,0)$ is the most likely trellis branch for $u_k = 0$ (bold, dashed line) and that $(s,s') = (2,1)$ is the most likely trellis branch for $u_k = 1$ (bold, solid line), $L_{\text{MLM}}(u_k)$ can be written as:

$$\begin{aligned} L_{\text{MLM}}(u_k) = &(A_k(2) + \Gamma_k(2,1) + B_{k+1}(1)) - \\ &(A_k(0,0) + \Gamma_k(0,0) + B_{k+1}(0)). \end{aligned} \quad (4)$$

On the other hand, the LLR provided by the MLM algorithm can also be seen in a different way, involving *paths* in the trellis diagram. Assuming that the trellis begins and ends at known states s_0 and s_K, then there are 2^K possible state sequences connecting s_0 and s_K. Each state sequence corresponds to a distinct input bit sequence and is associated with a *path* in the trellis. Given the received codeword $\mathbf{y}$, the path or state sequence $\mathbf{s}$ has a *path metric* equal to

$$\ln Pr(\mathbf{y}|\mathbf{s}) = \sum_{k=0}^{K-1} \ln Pr(y_k|s_k, s_{k+1}). \quad (5)$$

In other words, for a given path going through a series of branches, the path metric is the sum of all the involved branch metrics. The Viterbi algorithm (VA) [18] employs a recursive selection of paths in the trellis from state s_0 to state s_K in order to find the state sequence with the highest path metric and provides the associated bit sequence as the output of the decoder. In fact, the forward propagation of the MLM algorithm is identical to the path recursion in the VA, as already reported in [19]. The backward propagation from state s_K to state s_0 could equally be used for the same purpose.

For a binary convolutional code with rate $r = 1/m$, there are two branches in the trellis arriving at each state s_k at time index k. In the (forward) path recursion, the VA adds each of the two branch metrics to the path metric associated with the preceding state at time index $k-1$. It then selects the path with the larger metric as the *surviving path* and stores it. Therefore, at time index k, there are 2^ν surviving paths left, one for each state. Similarly, using the backward propagation, one can identify another set of 2^ν surviving paths at each time index and, in particular, at time index $k+1$.

Furthermore, thanks to the path convergence property of convolutional codes [18], the surviving paths can be truncated to some manageable length L to limit the amount of memory necessary for their storage, without any noticeable impact on the error correction performance. For a truncation length of L trellis sections, the VA needs to process the trellis paths until time index k to take the decision on the information bit at time index $(k-L)$. The value of L should be chosen large enough so that the 2^ν surviving paths from the forward propagation originate from a single state at time index $(k-L)$ with sufficiently high probability [20]. The same rule applies if a backward propagation is carried out: in this case, the VA needs to process the trellis paths until time index k' to take the decision on the information bit at time index $(k'+L)$.

Now, let us consider that both forward and backward propagations are carried out through the trellis. Then, for trellis section k, there are 2^ν surviving paths at time index k resulting from the forward propagation and 2^ν surviving paths at time index $k+1$ resulting from the backward propagation.

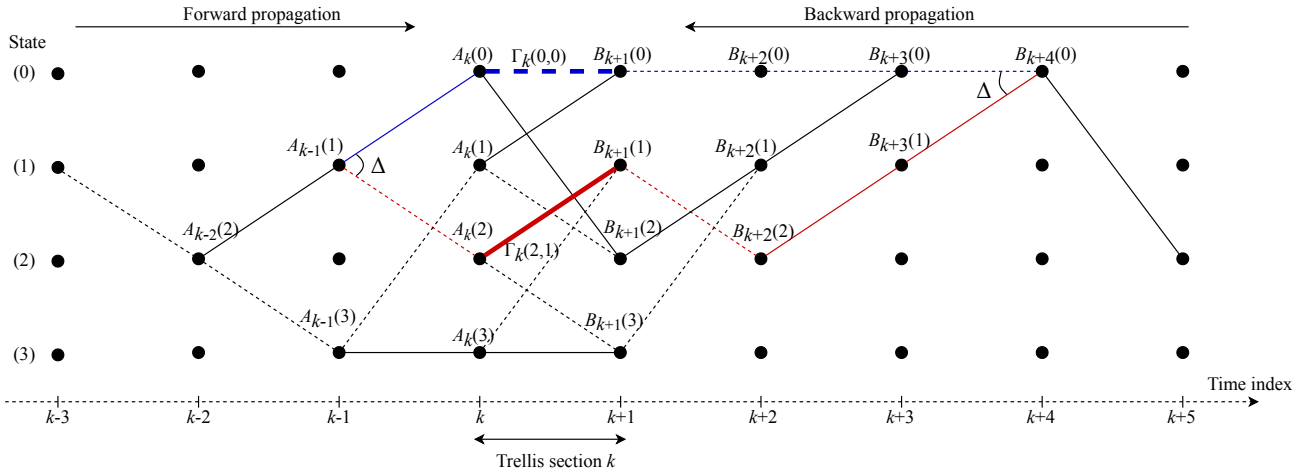

Fig. 1. Trellis representation of a convolutional code with $\nu = 2$. Dashed branches correspond to data bits equal to 0 and solid branches to data bits equal to 1.

Connecting these two sets of paths with the $2^{\nu+1}$ trellis branches yields a total number of $2^{\nu+1}$ surviving paths for trellis section k. The path going through state s at time index k and through state s' at time index $k+1$ has its path metric equal to $A_k(s) + \Gamma_k(s, s') + B_{k+1}(s')$. Moreover, because of the path convergence property, all $2^{\nu+1}$ paths originate from a single state at time index k' such that $\max(0, k-L) \leq k' < k$ in the forward propagation and from a single state at time index k'' such that $k+1 < k'' \leq \min(k+L+1, K)$ in the backward propagation. So, these $2^{\nu+1}$ paths merge together at times indexes k' and k''.

Many readers will recognize the concept of path merging exploited in the SOVA, [13]. Therefore, for each pair of merging paths, we can obtain the metric difference between these two paths by subtracting the lower path metric from the higher one. Then, we can manipulate these resulting metric differences between paths to get the soft estimate of bit u_k for trellis section k.

Taking the example in Fig. 1, the forward propagation and the backward propagation provide us with four forward surviving paths at time index k and four backward surviving paths at time index $k+1$, respectively. So, eight paths have to be considered for trellis section k, and as shown in Fig. 1, these paths merge in state 2 at time index $k' = k-2$ and in state 0 at time index $k'' = k+4$. Furthermore, if we denote by P_0 the path going through the branch (0,0) and by P_1 the path going through the branch (2,1) in trellis section k, the LLR of bit u_k can be estimated as the metric difference Δ between these two paths where the path metric M_0 of P_0 is $M_0 = (A_k(0) + \Gamma_k(0,0) + B_{k+1}(0))$ and the path metric M_1 of P_1 is $M_1 = (A_k(2) + \Gamma_k(2,1) + B_{k+1}(1))$. The result is equal to the LLR estimated by the MLM algorithm in (4).

Based on this equivalence, we can reformulate the MLM algorithm. In the trellis diagram, each trellis section k involves $2^{\nu+1}$ paths going through $2^{\nu+1}$ branches and merging together. If a path goes through the branch (s, s'), its path metric M_k is then expressed as

$$M_k = A_k(s) + \Gamma_k(s, s') + B_{k+1}(s'). \tag{6}$$

The soft output related to bit u_k is then equal to the metric difference between the path with the largest path metric carrying $u_k = 1$ and the path with the largest path metric carrying $u_k = 0$. In the next section, we define the path merge operation, present its properties and show how it can be used to reformulate the MLM algorithm.

III. The Local 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. From this section, we will adopt an alternative mathematical definition of a path, with a more local sense, that focuses on a particular trellis section k. In the following, all the derivations focus on trellis section k. Therefore, for sake of simplicity, we will omit k in the notations.

We define a path P as a 3-tuple consisting of a path metric denoted by M, a hard decision denoted by u and a reliability value related to u, denoted by L:

$$P = \{M, u, L\} \in \mathbb{R} \times \{0, 1\} \times \mathbb{R}^+, \tag{7}$$

where $\mathbb{R}$ is the set of real numbers and $\mathbb{R}^+$ is the set of positive real numbers.

As stated in the previous section, if path P goes through branch (s, s') at trellis section k, its path metric M is given by (6). Moreover, the hard decision u is the data bit carried by the corresponding branch in the trellis diagram (0 for a dashed line or 1 for a solid line in Fig. 1). The reliability of the hard decision, L, is initialized to $+\infty$ or to the largest possible value achievable with the used quantization.

We further define the *merge operation*

$$\mathcal{M} \ : \ \{\mathbb{R} \times \{0, 1\} \times \mathbb{R}^+\}^2 \rightarrow \mathbb{R} \times \{0, 1\} \times \mathbb{R}^+, \tag{8}$$

taking two paths as arguments and producing one path as output. $P_a = \{M_a, u_a, L_a\}$ and $P_b = \{M_b, u_b, L_b\}$ being two paths, determining path P_c such that $P_c = \mathcal{M}(P_a, P_b)$ involves three procedures: finding M_c, u_c and L_c. The output path metric M_c and hard decision u_c can be obtained by comparing

the path metrics M_a and M_b. Let $p = \arg\max_{a,b}(M_a, M_b)$, then $M_c = M_p$ and $u_c = u_p$. Through that mechanism, if several paths merge at a trellis stage, the resulting output path will be assigned the largest path metric and will be considered as the *maximum likelihood* (ML) path. The hard decision carried by the ML path is also the hard decision provided by the decoder. Furthermore, in order to find L_c, we employ two well-known reliability update rules: the Hagenauer rule (HR) [13] and the Battail rule (BR) [16], [17]. Both rules were proposed independently in the late 80's for SOVA decoders.

A. Reliability Update Rules

Let P_a and P_b be two paths to be merged and let us define p and p' as

$$p = \arg\max_{a,b}(M_a, M_b); \quad p' = \arg\min_{a,b}(M_a, M_b), \tag{9}$$

and the metric difference between P_a and P_b as $\Delta_{p,p'} = M_p - M_{p'}$. Note that the metric difference between two paths is always positive. Then, if $P_c = \mathcal{M}(P_a, P_b)$, L_c is calculated as follows:

1) If $u_a \neq u_b$, apply HR
$$L_c = \min\left(L_p, \Delta_{p,p'}\right) \tag{10}$$
2) If $u_a = u_b$, apply BR
$$L_c = \min\left(L_p, \Delta_{p,p'} + L_{p'}\right) \tag{11}$$

These two update rules can be summarized using the following update function ϕ:

$$\begin{aligned} L_c &= \phi\big(L_p, L_{p'}, \Delta_{p,p'}, u_p, u_{p'}\big) \\ &= \min\left(L_p, \Delta_{p,p'} + \delta(u_p, u_{p'})L_{p'}\right) \end{aligned} \tag{12}$$

where

$$\delta(u_p, u_{p'}) = \begin{cases} 1, & \text{if } u_p = u_{p'} \\ 0, & \text{otherwise.} \end{cases}$$

The combination of these two rules for SOVA decoding was already proposed in [21] and the authors proved the equivalence with the MLM algorithm. However, in [21], the authors only considered forward propagation. To estimate the reliability of the hardware decision at trellis section k, the algorithm carries out a forward propagation up to trellis stage $k + L$ and then performs a traceback procedure. Thus, a large number of paths should be considered, which translates into a massive use of function ϕ and also into large memory for storing the reliability values after each update.

We propose an alternative algorithm that uses both forward propagation and backward propagation and hence limits the number of paths considered for trellis section k to $2^{\nu+1}$, thus reducing the use of function ϕ.

B. Commutative and associative properties of the merge operation

As already mentioned earlier, the merge path operation described above involves three procedures: 1) selecting the output path metric 2) selecting the related hard decision and 3) updating the related reliability value using function ϕ in (12). We show in this section that the merge operation has the commutative and associative properties.

Theorem 1. *The merge operation is commutative and associative.*

- *Commutative property*
  Let P_a, P_b be two merging paths:
$$\mathcal{M}(P_a, P_b) = \mathcal{M}(P_b, P_a), \tag{13}$$
- *Associative property*
  Let P_a, P_b, and P_c be three merging paths:
$$\mathcal{M}(\mathcal{M}(P_a, P_b), P_c) = \mathcal{M}(P_a, \mathcal{M}(P_b, P_c)). \tag{14}$$

The "=" operator between two paths is defined as the equality between all the elements in their tuples.

Proof. We will prove that the three above-mentioned procedures are commutative and associative.

For the commutative property, let us define p and p' as in (9). The path metric of the output path is then M_p, the hard decision is u_p and according to (12), the reliability value is then equal to

$$\min(L_p, \delta(u_p, u_{p'})L_{p'} + \Delta_{p,p'}). \tag{15}$$

Since p and p' do not depend on the order of P_a and P_b, the merge operation is commutative.

Similarly, we can easily show that selecting a path metric and providing a hard decision are associative procedures because they get the values of the path with the largest metric and the maximum function is associative since

$$\max(M_a, \max(M_b, M_c)) = \max(\max(M_a, M_b), M_c).$$

Concerning the reliability update procedure, without loss of generality, we assume that M_c is the largest path metric and we define p and p' as in (9). The updated reliability values of $\mathcal{M}(P_a, \mathcal{M}(P_b, P_c))$ and $\mathcal{M}(\mathcal{M}(P_a, P_b), P_c)$ in (14) are respectively derived in (16) and (17) using function ϕ. The proof is then divided into two parts, corresponding to two cases: 1) $u_c = u_p$ and 2) $u_c \neq u_p$.

1) $\delta(u_c, u_p) = 1$, then (17) becomes:
$$\min\left(L_c, \delta(u_c, u_p)L_p + \Delta_{c,p}, \delta(u_c, u_{p'})L_{p'} + \Delta_{c,p'}\right). \tag{18}$$
Since $\Delta_{c,p'} = \Delta_{c,p} + \Delta_{p,p'}$, (18) coincides with (16).
2) $\delta(u_c, u_p) = 0$, then (17) becomes:
$$\min(L_c, \Delta_{c,p}). \tag{19}$$
Assuming $p = b$, (16) becomes:
$$\min\left(L_c, \Delta_{c,b}, \delta(u_c, u_a)L_a + \Delta_{c,b} + \Delta_{b,a}\right). \tag{20}$$
Since L_a and $\Delta_{b,a}$ are always positive, (20) becomes $\min(L_c, \Delta_{c,b})$ which coincides with (19).

Therefore, the associative property is proved for both cases. □

Remark. Based on the commutative and associative properties of the merge operation, two important statements can be inferred:

$$L_{\mathcal{M}(P_a,\mathcal{M}(P_b,P_c))} = \min\Big(\min\big(L_c, \delta(u_c,u_b)L_b + \Delta_{c,b}\big), \delta(u_c,u_a)L_a + \Delta_{c,a}\Big)$$
$$= \min\big(L_c, \delta(u_c,u_b)L_b + \Delta_{c,b}, \delta(u_c,u_a)L_a + \Delta_{c,a}\big), \tag{16}$$
$$L_{\mathcal{M}(\mathcal{M}(P_a,P_b),P_c)} = \min\Big(L_c, \delta(u_c,u_p)\min\big(L_p, \delta(u_p,u_{p'})L_{p'} + \Delta_{p,p'}\big) + \Delta_{c,p}\Big)$$
$$= \min(L_c, \delta(u_c,u_p)L_p + \Delta_{c,p}, \delta(u_c,u_p)\delta(u_p,u_{p'})L_{p'} + \delta(u_c,u_p)\Delta_{p,p'} + \Delta_{c,p}). \tag{17}$$

- We can extend the merge operation to more than two paths. For instance, for four paths P_a, P_b, P_c and P_d, we can write $\mathcal{M}(P_a, P_b, P_c, P_d)$ to refer to the output path obtained by merging the four paths.
- The merge operation can be processed in a *dichotomous* fashion:

  $$\mathcal{M}(P_a, P_b, P_c, P_d) = \mathcal{M}\big(\mathcal{M}(P_a, P_b), \mathcal{M}(P_d, P_c)\big),$$

  where $\mathcal{M}(P_a, P_b)$ and $\mathcal{M}(P_c, P_d)$ can be processed in parallel and then the resulting paths are merged to yield the output path.

C. The MLM algorithm as an instance of the merge operation

Referring back to Section II, let us consider the $2^{\nu+1}$ paths going through trellis stage k and merging together. Among them, $n = 2^{\nu}$ paths, denoted by $\{P_{p_1^0}, P_{p_2^0}, \ldots, P_{p_n^0}\}$, carry hard decision $u = 0$ at trellis section k and the remaining n, denoted by $\{P_{p_1^1}, P_{p_2^1}, \ldots, P_{p_n^1}\}$, carry hard decision $u = 1$. The reliability value related to bit u provided by the MLM algorithm is

$$L_{\text{MLM}}(u) = \max_{i=1,\ldots,n}\{M_{p_i^1}\} - \max_{j=1,\ldots,n}\{M_{p_j^0}\}. \tag{21}$$

Let us consider the operation merging all the paths with hard decision $u = 1$: $\mathcal{M}(P_{p_1^1}, \ldots, P_{p_n^1})$. The resulting output hard decision is obviously 1 and the output path metric is $M_1 = \max_{i=1,\ldots,n}\{M_{p_i^1}\}$. For the output reliability, merging paths with the same hard decision value requires the application of BR (11). Since the reliability values are all initialized at $+\infty$, applying (11) yields an output reliability also equal to $+\infty$. Similarly, merging together all the paths carrying hard decision $u = 0$ applying $\mathcal{M}(P_{p_1^0}, \ldots, P_{p_n^0})$ results in an output hard decision 0, an output path metric equal to $M_0 = \max_{i=1,\ldots,n}\{M_{p_i^0}\}$ and an output reliability at $+\infty$.

Then, if we merge the two resulting paths:

$$\mathcal{M}\big(\mathcal{M}(P_{p_1^1}, \ldots, P_{p_n^1}), \mathcal{M}(P_{p_1^0}, \ldots, P_{p_n^0})\big), \tag{22}$$

the computation of the output reliability amounts to the application of HR (10):

$$L_{\mathcal{M}} = \min\Big(+\infty, \Big|M_1 - M_0\Big|\Big) = \Big|M_1 - M_0\Big|, \tag{23}$$

which is the absolute value of the expression of L_{MLM} in (21). If we denote by $u_{\mathcal{M}}$ the output hard decision deriving from (22), then L_{MLM} is equal to

$$L_{\text{MLM}}(u_k) = (2u_{\mathcal{M}} - 1) \times L_{\mathcal{M}}. \tag{24}$$

Therefore, the result of the MLM algorithm can be interpreted as the outcome of a merge operation applied to all the paths.

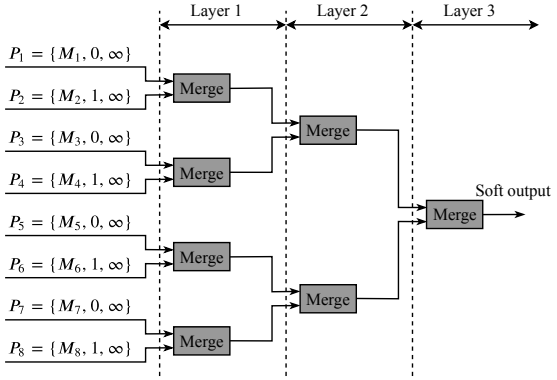

Fig. 2. Binary tree architecture used for the soft output computation in the local SOVA decoder for the code of Fig.1.

On another note, thanks to the commutative and associative properties of the merge operation stated in (13) and (14), the operation merging all the paths (22) can be performed in a different order for a better match with efficient software or hardware implementations. In particular, instead of first merging the paths with the same hard decision, one can start by merging pairs of paths with different hard decisions $(P_{p_i^1}, P_{p_i^0})$, $i = 1, \ldots, n$:

$$\mathcal{M}\big(\mathcal{M}(P_{p_1^1}, P_{p_1^0}), \ldots, \mathcal{M}(P_{p_n^1}, P_{p_n^0})\big). \tag{25}$$

Moreover, if $P_{p_i^1}$ and $P_{p_i^0}$ are chosen in such a way that the corresponding trellis branches at trellis section k, $(s_{p_i^1}, s'_{p_i^1})$ and $(s_{p_i^0}, s'_{p_i^0})$ verify $s'_{p_i^1} = s'_{p_i^0} = s'$, the merge operation of the pair of paths yields the following path metric:

$$M_{p_i} = \max(M_{p_i^1}, M_{p_i^0}) \tag{26}$$
$$= \max\big(A_k(s_{p_i^1}) + \Gamma_k(s_{p_i^1}, s') + B_{k+1}(s'),$$
$$A_k(s_{p_i^0}) + \Gamma_k(s_{p_i^0}, s') + B_{k+1}(s')\big)$$
$$= \max\big(A_k(s_{p_i^1}) + \Gamma_k(s_{p_i^1}, s'), A_k(s_{p_i^0}) + \Gamma_k(s_{p_i^0}, s')\big)$$
$$+ B_{k+1}(s') \tag{27}$$
$$= A_{k+1}(s') + B_{k+1}(s'), \tag{28}$$

and since $u_{p_i^1} \neq u_{p_i^0}$, the updated reliability with HR is

$$L_{p_i} = \min\big(+\infty, \Delta_{p_i^1,p_i^0}\big) = \Delta_{p_i^1,p_i^0} \tag{29}$$

where

$$\Delta_{p_i^1,p_i^0} = \Big|M_{p_i^1} - M_{p_i^0}\Big|$$
$$= \Big|\big(A_k(s_{p_i^1}) + \Gamma_k(s_{p_i^1}, s')\big) -$$
$$\big(A_k(s_{p_i^0}) + \Gamma_k(s_{p_i^0}, s')\big)\Big|. \tag{30}$$

The output hard decision u_{p_i} is provided by the path, $P_{p_i^0}$ or $P_{p_i^1}$, with the higher path metric.

We can see from (27) and (28) that, with the proposed merge ordering, the calculation of the output path metric for this scheduling proposal $\mathcal{M}(P_{p_i^1}, P_{p_i^0})$ includes the derivation of the forward state metric $A_{k+1}(s')$ as in the forward recursion (3a). Therefore, there is no need to perform a preliminary calculation of the forward state metrics. Only the backward state metrics need to be computed in advance. Similarly, one can show that if the paths $P_{p_i^1}$ and $P_{p_i^0}$ are chosen in such a way that the corresponding trellis branches at trellis section k are stemming from the same state ($s_{p_i^1} = s_{p_i^0} = s$), the calculation of the output path metric includes the same derivation of the backward state metric $B_k(s)$ as in the backward recursion (3b). Then, there is no need to perform a preliminary calculation of the backward state metrics and only the forward state metrics need to be computed in advance.

For a convolutional code with memory length ν, the overall merge operation for the computation of the soft output for the decoder can be carried out in a dichotomous fashion : the merge operation then requires a binary tree of $2^{\nu+1}-1$ elementary merge operators organized in $\nu+1$ layers. Taking Fig. 1 as an example, $\nu = 2$ and the soft output related to bit u_k is obtained by a binary tree consisting of $\nu+1 = 3$ layers of merge operators, as shown in Fig. 2.

The next section describes the overall algorithm implementing this particular arrangement for the merge operations.

D. Soft output computation algorithm

The soft output calculation in a dichotomous fashion can be performed according to Algorithm 1.

Note that this algorithm is a generic version that assumes that all the forward and backward state metrics were pre-computed and stored in a memory. In practice, different schedules can be applied for the recursive computation of the state metrics, including the well-known backward-forward (BF) and butterfly schedules [7], [22]. When compared, the former takes twice the time to decode a frame while the latter requires twice the hardware resources. Nevertheless, both these schedules have the same overall computational complexity. Therefore, for the rest of the paper, we only consider the BF schedule, the application of the results to other schedules being straightforward.

With the BF schedule, the backward state metrics are pre-computed and stored in a memory. Then, the algorithm recursively calculates the forward state metric and derives the soft output. As discussed in the previous section, specific simplifications can be made for the local SOVA at layer $l = 1$. First, at the initialization step, we can organize the set of transitions $\mathbf{T}$ so that two adjacent transitions have the same state s', i.e., $s'_{2i-1} = s'_{2i}$, for $i = 1, \ldots, 2^\nu$. Then, in line 5, we can initialize path metrics $M_p(0)$ to be $A_k(s) + \Gamma_k(s, s')$ instead of $A_k(s) + \Gamma_k(s, s') + B_{k+1}(s, s')$. The resulting path metric $M_a(1)$ at layer $l = 1$ is equal to $A_{k+1}(s')$, thus allowing the forward recursion to be incorporated into the soft output computation. To compensate for omitting $B_{k+1}(s')$ in the expression of $M_p(0)$, in line 14, the output path metric $M_p(1)$ should be taken equal to

$$M_p(1) = M_a(1) + B_{k+1}(s'). \tag{31}$$

Furthermore, in line 16, output reliability $L_p(1)$ can be directly assigned $\Delta_{a,b}$ and the calculation of the reliability value at the first layer can be replaced by a simple assignment operation, making the initial assignment in line 7 unnecessary. After layer $l = 1$, the subsequent layers should be carried out following Algorithm 1 without any modification.

To complete this algorithmic description, the next sections provide some details about possible hardware architectures for a local SOVA decoder. To this end, we first focus on a radix-2 trellis in Section III-E and highlight the differences with a conventional MLM decoder. Section IV will later describe higher radix architectures.

Algorithm 1 The local SOVA

1: **Initalization**: $\mathbf{T} = [T(1) \ \ldots \ T(2^{\nu+1})]$ is the set of $2^{\nu+1}$ transitions of the trellis section defined by the pairs of states (s, s');
2: **for** each trellis stage $k = 1, \ldots, K$ **do**
3: **for** each path $p = 1, \ldots, 2^{\nu+1}$ **do**
4: $(s_p, s'_p) = T(p)$
5: $M_p(0) = A_k(s_p) + \Gamma_k(s_p, s'_p) + B_{k+1}(s'_p)$;
6: $u_p(0) =$ data bit on transition (s_p, s'_p);
7: $L_p(0) = +\infty$;
8: **end for**
9: **for** each layer $l = 1, \ldots, \nu+1$ **do**
10: **for** each path $p = 1, \ldots, 2^{(\nu-l+1)}$ in layer l **do**
11: $a = \arg\max_{j\in\{2p-1,2p\}}\{M_j(l-1)\}$;
12: $b = \arg\min_{j\in\{2p-1,2p\}}\{M_j(l-1)\}$;
13: $\Delta_{a,b} = M_a(l-1) - M_b(l-1)$;
14: $M_p(l) = M_a(l-1)$;
15: $u_p(l) = u_a(l-1)$;
16: $L_p(l) = \phi\big(L_a(l-1), L_b(l-1), \Delta_{a,b}, u_a(l-1), u_b(l-1)\big)$;
17: **end for**
18: **end for**
19: Hard output: $\hat{u}_k = u_1(\nu+1)$;
20: Soft output: $\hat{L}(u_k) = (2\hat{u}_k - 1)\times L_1(\nu+1)$;
21: **end for**

E. Radix-2 local SOVA decoder architecture

Since the proposed algorithm only differs from the MLM algorithm in the soft output calculation, its global architecture is composed of the same blocks as the architecture of a MLM decoder [23]: in the case of a BF decoding schedule, *branch metric units* (BMU) and *add-compare-select units* (ACSU) recursively compute the backward and forward state matrics, a *state metric memory* stores the backward state metrics and a *soft-output unit* (SOU) computes the extrinsic information and the soft decision during the forward recursion. Fig. 3 shows the corresponding basic architecture for the forward recursion; the backward BMU and ACSU are not shown. If a symmetric forward-backward scheduling is applied, the roles of forward and backward units are just swapped.

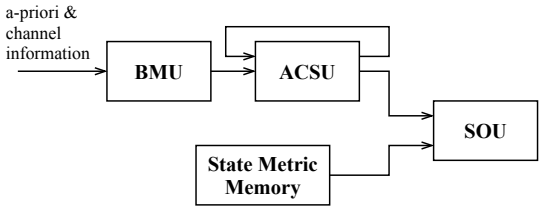

Fig. 3. Basic architecture considered for the local SOVA and MLM algorithms.

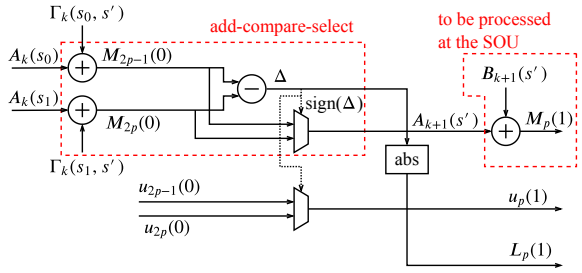

Fig. 4. Hardware architecture of Algorithm 1 for layer $l = 1$.

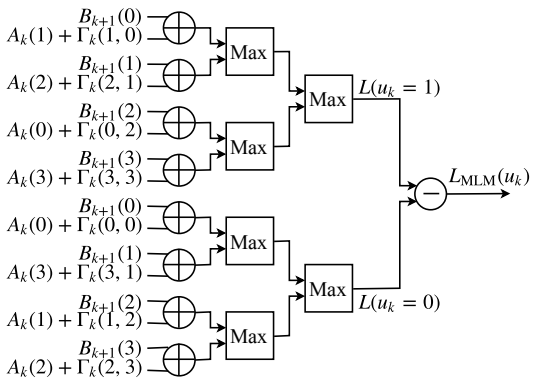

Fig. 5. SOU of a MLM decoder for $\nu = 2$.

At trellis stage k, the BMU calculates all possible values of $\Gamma_k(s, s')$ and forwards them to the ACSU. The ACSU calculates $A_k(s)+\Gamma_k(s, s')$ for each branch in the trellis stage and computes (3a) for each state s' at time index $k + 1$. For the local SOVA architecture, the first layer of Algorithm 1, depicted in Fig. 4, performs the forward state metric recursion as well as it produces the hard decision, the reliability value and the path metric for subsequent layers. Besides the right-hand side adder, the structure shown in Fig. 4 for $l = 1$ is very close to the ACSU structure of a conventional MLM decoder. Therefore, in order to make the local SOVA easy to compare with the MLM algorithm, we consider this substructure as the ACSU of the local SOVA decoder and the final adder plus the units processing the subsequent layers as its SOU.

The main difference between both architectures comes from the SOU. In the MLM architecture, $2^{\nu+1}$ intermediate values $A_k(s) + \Gamma_k(s, s')$ computed in the ACSU are added to $B_{k+1}(s')$. Then, the most reliable branch for bit $u_k = 1$ and for bit $u_k = 0$ are selected using two 2^ν-input maximum selection operators and the LLR is obtained by computing the

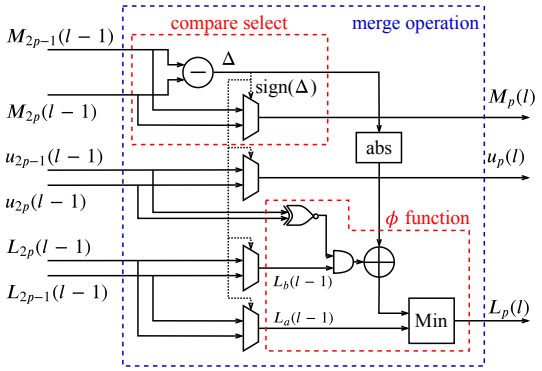

Fig. 6. Generic hardware architecture of a merge operation $\mathcal{M}$ of two paths indexed $2p$ and $2p - 1$ at layer $l > 1$.

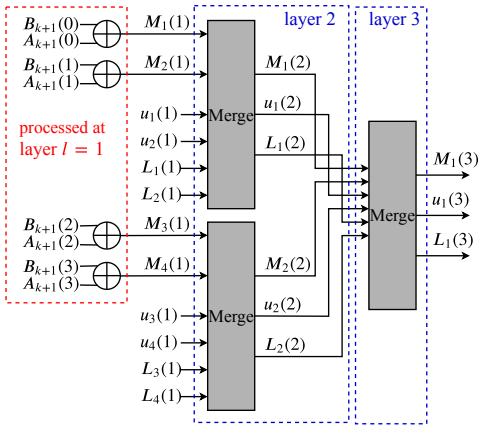

Fig. 7. SOU of a local SOVA decoder using merge operators for $\nu = 2$.

difference between the terms $A_k(s)+\Gamma_k(s, s')+B_{k+1}(s')$ for these two branches as shown in Fig. 5 for $\nu = 2$. On the other hand, the local SOVA SOU takes the 2^ν values of $A_{k+1}(s')$, adds them to 2^ν corresponding values of $B_{k+1}(s')$ to provide the path metrics. The hard decisions and their reliability values computed by the ACSU are forwarded to the SOU. The SOU has then to process 2^ν paths using a binary tree of merge operators. The structure of merge operators used to process layers $l > 1$ is depicted in Fig. 6 and the overall structure of the tree is shown in Fig. 7 for $\nu = 2$ (including the adders actually part of layer $l = 1$).

In terms of computational complexity, with the convention that one adder is equivalent to one max or min operator and is counted as one computational unit and neglecting the multiplexers, the operator implementing function ϕ consists of one adder and one min operator and is therefore counted as two computational units. Consequently, the MLM SOU requires $(4 \times 2^\nu - 1)$ computational units while the local SOVA SOU requires $(4\times 2^\nu - 3)$ computational units. Since the ACSU and the BMU of both architectures are similar, both algorithms have roughly the same computational complexity when a conventional radix-2 architecture is implemented. However, the proposed algorithm is mainly of interest when higher radix orders are considered, as explained in the following section.

IV. High-Radix Decoder Architectures using Local SOVA

In the previous section, we have been considering the conventional radix-2 trellis diagram. In this section, we will concentrate on higher radix orders. In a radix-2^R trellis diagram, $R \in \mathbb{N}^*$, there are 2^R branches coming in and out of a state s at time index k. This is obtained by aggregating R consecutive radix-2 trellis stages. Hence, a branch in a radix-2^R trellis stage is now labeled with R systematic bits and we have to reconsider the definition of a path and its corresponding merge operation.

For a radix-2^R trellis diagram, we define a radix-2^R path P going through the branch (s, s') as

$$P = \{M, u^1, \ldots, u^R, L^1, \ldots, L^R\} \in \mathbb{R} \times \{0,1\}^R \times \{\mathbb{R}^+\}^R \tag{32}$$

where M is the path metric, $u^1, \ldots, u^R$ are the R hard decisions attached to branch (s, s'), and $L^1, \ldots, L^R$ are the reliability values for each hard decision and are initialized to $+\infty$.

We also define the radix-2^R merge operation $\mathcal{M}$ as in Section III with three procedures: path metric selection, hard decision selection and update of the reliability values. The selection of the path metric remains unchanged. The only difference is now that we have to select R hard decisions instead of one, and to update R reliability values using function ϕ, one for each hard decision. Note that the merge operation for high-radix paths is also commutative and associative, therefore, the order of the paths in the merge operation does not affect the output. To this end, if we arrange wisely the input paths, we can reduce complexity when compared to a straightforward implementation.

A. Radix-4 local SOVA decoder with minimum complexity ACSU

A branch in a radix-4 trellis diagram is the aggregation of two consecutive branches in a radix-2 diagram, as illustrated in Fig. 8 for an convolutional code with $\nu = 2$. From time index k to time index $k+2$, four branches are leaving and merging into each trellis state, corresponding to the transmission of two systematic bits with possible values 00, 10, 01 and 11. Therefore, at time index $k+2$, there are four radix-4 paths, denoted by $\{P_{00}, P_{01}, P_{10}, P_{11}\}$, merging into each state s'. Since these four paths have the same $B_{k+2}(s')$ value, as in section III-D we can initialize the corresponding path metrics with $A_k(s) + \Gamma_{k\to k+1}(s, s')$ instead of $A_k(s) + \Gamma_{k\to k+1}(s, s') + B_{k+2}(s')$, where $\Gamma_{k\to k+1}(s, s')$ is the sum of the two successive branch metrics at time indices k and $k+1$ in the equivalent radix-2 tellis diagram. Then, we perform the radix-4 merge operation:

$$\mathcal{M}\big(P_{00}, P_{01}, P_{10}, P_{11}\big). \tag{33}$$

The path metric resulting from this layer-1 merge operation is also the forward state metric of state s' at time index $k+2$, $A_{k+2}(s')$. Hence, we can consider the operator implementing (33) as the radix-4 ACSU of the local SOVA decoder.

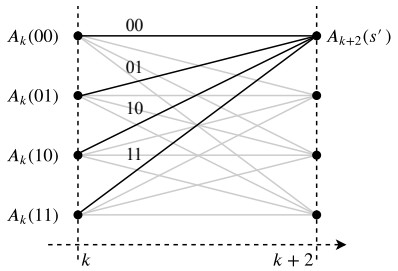

Fig. 8. A radix-4 stage for an convolutional code with $\nu = 2$.

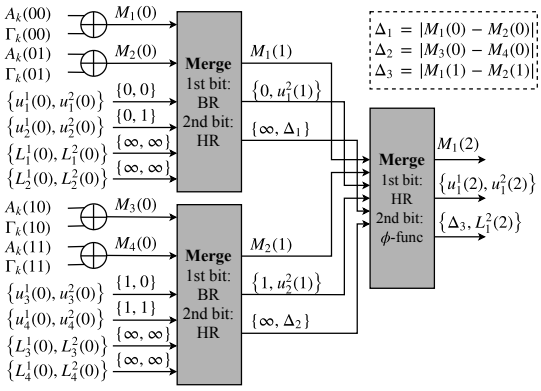

Fig. 9. Radix-4 local SOVA ACSU architecture implementing 2-bit merge operators according to (34).

An important property of the radix-4 local SOVA ACSU is that its complexity depends on the processing order of the paths in the merge operation (33). If we implement (33) as

$$\mathcal{M}\big(\mathcal{M}(P_{00}, P_{01}), \mathcal{M}(P_{10}, P_{11})\big) \tag{34}$$

with the hardware architecture shown in Fig. 9, only one ϕ operator has to be implemented. Actually, we do not need to resort to function ϕ in the first layer since the output reliability is $+\infty$ or the metric difference between the two paths, depending whether BR or HR is employed. On the other hand, if we implement (33) as

$$\mathcal{M}\big(\mathcal{M}(P_{00}, P_{11}), \mathcal{M}(P_{01}, P_{10})\big), \tag{35}$$

we can use HR for both bits at the first layer but have to use a ϕ operation for each bit at the second layer since we do not know *a priori* which hard decision will be selected. Therefore, the implementation of (34) is less complex than the one of (35) and it has minimum complexity. Note that (34) is not unique, since other processing orders of the paths can also yield the same complexity, such as $\mathcal{M}\big(\mathcal{M}(P_{00}, P_{10}), \mathcal{M}(P_{01}, P_{11})\big)$.

At the output of the radix-4 ACSU, 2^ν radix-4 paths are forwarded to the radix-4 SOU. First of all, each path metric is added to the appropriate $B_{k+2}(s')$ since it has been omitted in the ACSU. Then, a ν-layer tree of radix-4 merge operators is employed to produce the final hard decision along with its reliability value. Note that the radix-4 merge operation requires two ϕ operators for updating the reliability values, one for each bit.

In terms of computational complexity, the radix-4 local SOVA ACSU with minimum complexity requires one additional ϕ operator compared to the radix-4 MLM ACSU. Using the same convention as in Section III-E, processing the 2^ν states of the trellis then costs $2^{\nu+1}$ extra computational units. In return, the SOU is less complex in the case of the local SOVA. The SOU of the MLM decoder takes $4 \times 2^\nu$ intermediate values of $A_k(s)+\Gamma_{k\to k+1}(s,s')$ and adds them to the $B_{k+2}(s')$ values, using $4\times 2^\nu$ adders. Next, two maximum selection trees, each consisting of $(2 \times 2^\nu - 1)$ max operators, and a subtractor are needed for each systematic bit. Each LLR value is then obtained by using one extra subtractor, leading to a total computational complexity of $(12\times 2^\nu - 2)$ computational units for the SOU of the radix-4 MLM algorithm. In contrast, in a radix-4 local SOVA SOU, the values of $B_{k+2}(s')$ are added to 2^ν path metrics using 2^ν adders. Then, the ν-layer tree of radix-4 merge operators consists of $(2^\nu - 1)$ max operators and $2 \times (2^\nu - 1)$ ϕ operators. Therefore, the total complexity of the radix-4 Local SOVA SOU is $(6 \times 2^\nu - 5)$ computational units, which is less than half the complexity of the MLM algorithm SOU. All taken into account, the complexity of the local SOVA is reduced by approximately $4 \times 2^\nu$ computational units compared to the MLM algorithm.

When radix orders higher than 4 are considered, techniques for further reducing complexity can be considered, as explained in the following section.

B. Radix-8 local SOVA decoder using a simplified reliability update operator and application to the LTE turbo code

In this section, we introduce a sub-optimal but less complex version of the function ϕ, called ω. Consequently, by combining the corresponding operator with the minimum complexity ACSU previously described in Section III-E, we propose a number of local SOVA architectures with different complexity-performance tradeoffs.

A radix-8 trellis stage aggregates three consecutive radix-2 trellis stages. Each path is now composed of a path metric, three hard decisions and their three reliability values. From time index k to time index $k+3$, eight radix-8 branches are leaving and merging into each trellis state. Therefore, at time index $k+3$, there are eight radix-8 paths merging into each state s', denoted by $\{P_{000}, P_{001}, \ldots, P_{111}\}$, where the indices represent the hard decisions associated with each path. Similarly to the previous cases, when applying the radix-8 merge operation to this set of paths

$$\mathcal{M}\big(P_{000}, P_{001}, \ldots, P_{111}\big), \tag{36}$$

the resulting path metric is also the forward state metric of state s' at time index $k+3$, $A_{k+3}(s')$. This merging step can therefore be considered as the ACSU of the radix-8 local SOVA decoder. Then the output path is processed by the SOU to produce the soft decision.

The minimum complexity radix-8 ACSU can be obtained by implementing (36) as

$$\begin{aligned}\mathcal{M}\Big(&\mathcal{M}\big(\mathcal{M}(P_{000},P_{001}),\mathcal{M}(P_{010},P_{011})\big),\\ &\mathcal{M}\big(\mathcal{M}(P_{100},P_{101}),\mathcal{M}(P_{110},P_{111})\big)\Big).\end{aligned} \tag{37}$$

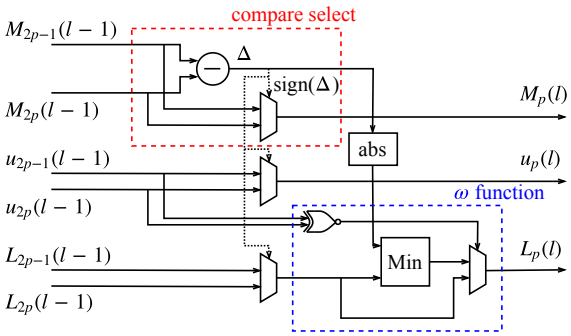

Fig. 10. Generic hardware architecture of path merging operation using function ω.

We can see from (37) that the first bit in $\mathcal{M}\big(\mathcal{M}(P_{000},P_{001}),\mathcal{M}(P_{010},P_{011})\big)$ is always zero, hence, we can resort to a radix-4 merge operation for the last two bits using only one ϕ operator. Similarly, the first bit in $\mathcal{M}\big(\mathcal{M}(P_{100},P_{101}),\mathcal{M}(P_{110},P_{111})\big)$ is always one, then again, only one ϕ operator is necessary. In the second stage, the first bit is always different in the two input paths, thus only the second and the third bits require the implementation of a ϕ operator. In total, four ϕ operators have to be implemented in a minimum complexity radix-8 ACSU.

However, employing ϕ operators has two main drawbacks. First of all, they consist of one adder and one min operator, therefore prohibitively increasing the complexity of the decoder if used excessively. Second and more importantly, the adder and the min operators are connected serially. This is not desirable since the ACSU dictates the critical path of the decoder [23]. Therefore, we propose a lower complexity, lower latency, sub-optimal update operator, based on new update function, called function ω.

1) The function ω: motivated by the the possibility of using only HR as in [13], one can substitute a simplified function ω for function ϕ. Assuming that paths P_{2p-1} and P_{2p} are to be merged at layer $l-1$, the output reliability is then computed as:

$$\begin{aligned}L_p(l) &= \omega\big(L_a, \Delta_{a,b}, u_{2p-1}(l-1), u_{2p}(l-1)\big)\\ &= \begin{cases}\min(L_a,\Delta_{a,b}), & \text{if } u_{2p-1}(l-1)\neq u_{2p}(l-1)\\ L_p, & \text{if } u_{2p-1}(l-1)= u_{2p}(l-1)\end{cases}\end{aligned} \tag{38}$$

where a, b and $\Delta_{a,b}$ are defined at lines 11, 12 and 13, respectively, in Algorithm 1. Fig. 10 shows the architecture of the elementary path merging operator using function ω. The ω operator is less complex than the ϕ operator since it uses only a min operation and a multiplexer resulting in a computational complexity of about one unit. However, the price to pay is a degradation of error correction performance. Indeed, a performance degradation of 0.5 dB is observed between the conventional SOVA that uses only functions ϕ [21] and the one that uses functions ω [13].

Unlike the conventional SOVA, local SOVA can mix both types of functions. Therefore, this provides the flexibility of several complexity/correction performance trade-offs. However, care must be taken in making substitutions so that paths

with high metrics are not eliminated from the selection process due to simplification. This is less likely to happen if the simplifications are made in the first layers of the tree, where the number of paths to be processed is high. Consequently, we observed that if we only substitute the ω operators for the ϕ operators in the first layers, we can significantly reduce complexity without degrading the performance of the decoder.

2) Radix-8 ACSU and SOU using ω operators: for a binary convolutional code with $\nu = 3$, the first 3 layers of the path merge binary tree are in the ACSU while the last 3 layers are processed by the SOU.

As already mentioned above, the radix-8 ACSU requires 4 ϕ operators to update the reliability values. Substituting the ω operators for the ϕ operators reduces the complexity by 4 computational units. For $\nu = 3$, the use of $2^{\nu} = 8$ ACSUs saves 32 computation units.

For the 3 layers of the binary tree implementing the radix-8 SOU, 7 radix-8 merge operations are required, resulting in the use of 21 ϕ operators (3 per merge operation). Replacing ϕ operators with ω operators would reduce complexity but would certainly penalize the performance of the decoder.

3) Simulation results with the LTE turbo code: we performed simulations to assess the error correcting performance of the radix-8 local SOVA and its variants for a LTE turbo decoder and compare them with the radix-8 MLM algorithm. We use the notations ACSU-(i, j) and SOU-(i, j) to represent the different configurations, where i and j are the number of layers where ω and ϕ operators are employed, respectively. For example, ACSU-(2,1) means that ω operators and implemented in the two first layers of the ACSU and that ϕ operators are used in the last layer.

We simulated the following seven configurations for the SISO decoding algorithms, where L-SOVA is an abbreviation for local SOVA:

- DEC 1: MLM algorithm.
- DEC 2: L-SOVA with ACSU-(0,3) and SOU-(0,3).
- DEC 3: L-SOVA with ACSU-(3,0) and SOU-(0,3).
- DEC 4: L-SOVA with ACSU-(3,0) and SOU-(1,2).
- DEC 5: L-SOVA with ACSU-(3,0) and SOU-(2,1).
- DEC 6: L-SOVA with ACSU-(3,0) and SOU-(3,0).
- DEC 7: conventional SOVA.

Our first motivation is to validate the equivalence between the local SOVA with only ϕ operators and the MLM algorithm. Second, we gradually substitute ω operators for ϕ operators to observe the impact on error correction behavior.

The simulations were carried out with information frames of $K = 1056$ bits, encoded with the non-punctured $r = 1/3$ LTE turbo code, modulated by BPSK and transmitted over the AWGN channel. A floating point representation of data is used in the decoder. The resulting bit error rate (BER) is measured after 5.5 decoding iterations. Fig. 11 shows that, as expected, the local SOVA with only ϕ operators has the same performance as the MLM algorithm. Moreover, by substituting ω operators for ϕ operators in the ACSUs, i. e. in the first three layers, the simulated curves confirm that the error correction performance of the decoder is not degraded, thus providing a low complexity alternative to the original local SOVA decoder. By gradually replacing the ϕ operators in the SOU,

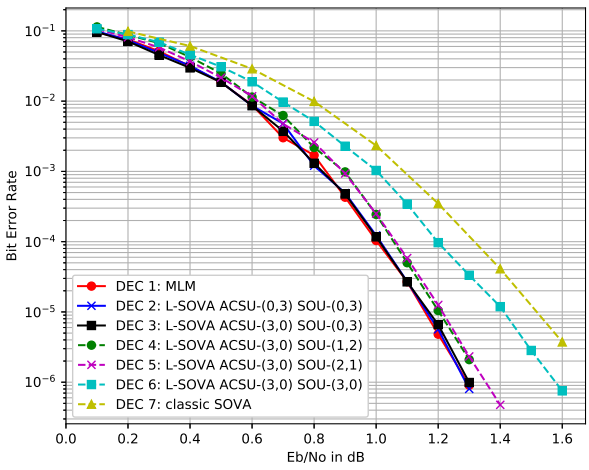

Fig. 11. BER performance of a LTE decoder using radix-8 MLM algorithm, local SOVA and its variant with $K = 1056$, $r = 1/3$ after 5.5 iterations. AWGN channel, BPSK modulation.

the performance is degraded by 0.05 dB at $BER = 10^{-6}$ when ω operators are used in the first two layers and by about 0.3 dB when only ω operators are implemented. Fig. 11 also shows that the conventional SOVA (DEC 7) performs 0.1 dB worse than the local SOVA with only ω operators (DEC 6). According to [21], using ϕ operators instead of ω operators in the both the conventional and local SOVAs would be equivalent to apply the MLM algorithm (DEC 1). However, contrary to the conventional SOVA, the operations in the local SOVA are arranged so as to minimize the number of used ϕ operators. Therefore the performance gap between DEC 6 and DEC 7 could be explained by the fact that the number of sub-optimal ω operators needed in DEC 7 is greater than in DEC 6.

4) Computational complexity analysis: The complexity of the considered decoding algorithm variants is shown in Table I. In this table, the decoding complexity (denoted by $\mathcal{C}$) is reported for one radix-8 trellis stage consisting of a backward ACSU, a forward ACSU and an SOU. For the MLM algorithm, the backward and forward ACSUs have same computational complexity. For the local SOVA and its variants, since we are considering the BF schedule, the backward ACSU is the same as in the MLM algorithm, but the forward ACSU is the newly developed one with ϕ and/or ω operators. Furthermore, we take the MLM decoding complexity as a reference for normalization. Since the RSC code of the LTE turbo code has memory length $\nu = 3$, the complexity of one radix-8 trellis stage can be approximated as

$$\mathcal{C} = 8 \times \mathcal{C}_{\text{backward ACSU}} + 8 \times \mathcal{C}_{\text{forward ACSU}} + \mathcal{C}_{\text{SOU}} \qquad (39)$$

We can observe that using local SOVA with only ϕ operators only amounts to 73% of the reference complexity while we can further reduce the cost to 67% without noticeable impact on the performance if we use ω instead of ϕ operators in the ACSUs. If a lower complexity is desired, the local SOVA with ACSU-(3,0) and SOU-(2,1) can be employed to reach

TABLE I
COMPARISON OF THE COMPUTATIONAL COMPLEXITY OF VARIOUS RADIX-8 ALGORITHMS (CS: COMPARE-SELECT).

Algorithm	8 × backward ACSU		8 × forward ACSU		SOU		Computational complexity $\mathcal{C}$	Normalization
	Adder	CS	Adder	CS	Adder	CS		
MLM	64	56	64	56	67	186	493	1
L-SOVA ACSU-(0,3), SOU-(0,3)	64	56	96	88	29	28	361	0.73
L-SOVA ACSU-(3,0), SOU-(0,3)	64	56	64	88	29	28	329	0.67
L-SOVA ACSU-(3,0), SOU-(1,2)	64	56	64	88	17	28	317	0.64
L-SOVA ACSU-(3,0), SOU-(2,1)	64	56	64	88	11	28	311	0.63
L-SOVA ACSU-(3,0), SOU-(3,0)	64	56	64	88	8	28	308	0.62

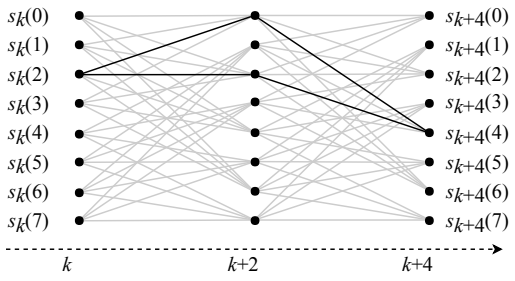

Fig. 12. A radix-16 trellis stage consisting of two radix-4 trellis stages.

63% of the reference complexity. This reduced complexity is then traded for a performance degradation of 0.05 dB. Table I also shows that using only ω operators further reduces the complexity by 1% but the resulting 0.3 dB performance loss makes this configuration less attractive.

C. Radix-16 local SOVA decoder for convolutional codes with memory length $\nu = 3$: breaking the ACSU bottleneck

In this section, we only consider more practical convolutional codes with $\nu = 3$ (8 states) to show the advantage of the local SOVA compared to the conventional MLM algorithm. A section of a radix-16 trellis diagram is the aggregation of four consecutive sections in a radix-2 trellis diagram or, equivalently, of two consecutive sections in a radix-4 trellis diagram. With the latter representation, two branches are connecting any pair of states s and s' between time indices k and $k+4$, as illustrated in Fig. 12 between $s = 2$ and $s' = 4$.

Intuitively, one could decode the radix-16 turbo code with a radix-16 ACSU using 16-input max-select operations. However, increasing the number of inputs of the max-select operations results in a longer propagation path in the ACSU thus lengthening the critical path of the decoder. To get around this problem, the authors of [12] suggested that we can select the branch with larger branch metric among the two branches connecting two states at time indices k and $k+4$, and discard the other one. Since this task could be done in the BMU to reduce by half the number of branches, we can then use radix-8 ACSUs instead for calculating the state metrics.

However, the main drawback of this approach when applied to the MLM algorithm is that the branches selected after the BMU might carry either $u_j = 1$ or $u_j = 0$, for $j = k, \ldots, k+3$. This creates a non-static ratio between the number of branches having $u_j = 1$ and the number of branches having $u_j = 0$, consequently causing a major problem for the MLM SOU since it naturally employs max-select operations with a constant number of inputs referring to hard decisions 1 and 0.

On the other hand, from the local SOVA perspective, the two branches processed by the BMU can be considered as two paths merging at time index $k+4$. Hence, we can use the path merging operation to produce one path carrying the selected hard decision and the updated reliability value. The BMU can then forward 64 paths into the radix-8 ACSUs and then the 8 output paths are sent to the SOU to finally compute the soft output. So, the BMU, the ACSU and the SOU constitute a path merging binary tree with 7 layers: the first layer is in the BMU, and thus not in the critical path, the three next layers are in the ACSU and the last three layers in the SOU. Also, because the order of the path inputs does not affect the result of the merge operation, the local SOVA is immune to the problem of the non-static ratio of hard decision value mentioned above for the MLM algorithm.

Extensive simulations for radix-16 local SOVA were also carried out. Similarly to the radix-8 case, we gradually substituted ω operators for ϕ operators in order to observe the behavior of the decoder. The results are shown in Fig. 13. We can observe that the performance equivalence between the MLM algorithm and the local SOVA with only ϕ operators still holds for the radix-16 case. Moreover, it is shown that we can still replace ϕ operators by ω operators in the ACSU and in the first layer of the SOU with a negligible degradation in performance. However, further substitutions are not recommended since a penalty of 0.4 dB at 10^{-4} BER can be observed if we use solely ω operators.

D. Convolutional codes with radix orders higher than 16

Similarly, for convolutional codes with 8 states, by employing higher radix orders such as 32 and 64, there are respectively 4 and 8 branches in parallel connecting two states in a trellis section. In this case, the BMU will select the path with the largest path metric among the 4 or 8 paths. Moreover, since the BMU does not have the recursive loop as in the ACSU, it could be pipelined to ensure that the critical path always resides in the ACSU. However, since the complexity of the decoder increases exponentially with the

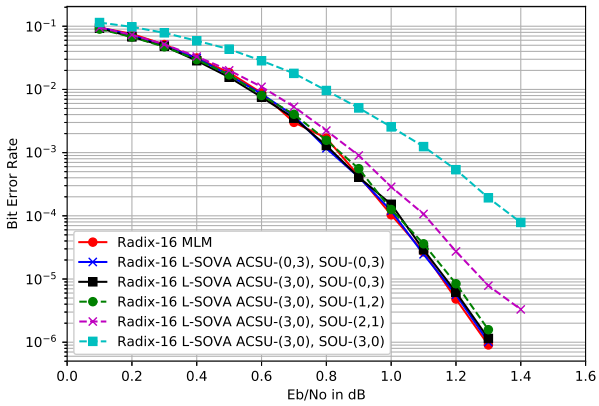

Fig. 13. BER performance of a LTE decoder using radix-16 MLM algorithm, local SOVA and its variant with $K = 1056$, $r = 1/3$ after 5.5 iterations. AWGN channel, BPSK modulation.

number of bits decoded simultaneously, it is necessary to find the processing order of the paths with the best compromise between throughput and complexity. Nonetheless, using local SOVA with high radix orders provides an ultra high throughput solution for turbo codes since the critical path of the decoder remains in the radix-8 ACSU while decoding an increasing number of systematic bits in a single clock cycle.

V. Conclusion

In this paper, we introduced a new SISO decoding algorithm for convolutional codes: local SOVA. The local SOVA architecture is shown to exhibit a more hierarchical structure and a lower computational complexity than the conventional Max-Log-MAP algorithm. We observed that using local SOVA in radix-8 LTE turbo decoders significantly reduces the complexity of the decoder compared to the respective radix-8 Max-Log-MAP architecture. Moreover, local SOVA makes it possible to increase the radix order without penalizing the error correction performance or the critical path of the decoder, at the price of added complexity. These advantages make local SOVA a first-choice algorithm for ultra-high throughput turbo decoders. Future work will include the investigation of very high radix orders, as well as implementations of merge operations for different radix orders.

Acknowledgement

This work was partially funded by the EPIC project of the EU's Horizon 2020 research and innovation programme under grant agreement No. 760150.

References

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

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

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

[4] C. Douillard and M. Jézéquel, "Chapter 1. Turbo codes: from first principles to recent standards," in *Channel Coding: Theory, Algorithms, and Applications*. Academic Press, July 2014, pp. 1–53.

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *JPL TDA Progr. Rep.*, vol. 42, no. 127, pp. 1–20, 1996.

[6] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb 2005.

[7] O. Muller, A. Baghdadi, and M. Jézéquel, "Exploring parallel processing levels for convolutional turbo decoding," in *Proc. 2nd ICTTA Conf.*, April 2006, pp. 2353–2358.

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

[9] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck," *IEEE Trans. Commun.*, vol. 37, no. 8, pp. 785–790, Aug 1989.

[10] C. Tang, C. Wong, C. Chen, C. Lin, and H. Chang, "A 952ms/s Max-Log MAP decoder chip using radix-4× 4 ACS architecture," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov 2006, pp. 79–82.

[11] K. Shr, Y. Chang, C. Lin, and Y. Huang, "A 6.6pj/bit/iter radix-16 modified log-MAP decoder using two-stage ACS architecture," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov 2011, pp. 313–316.

[12] O. Sánchez, C. Jégo, M. Jézéquel, and Y. Saouter, "High speed low complexity radix-16 Max-Log-MAP SISO decoder," in *Proc. IEEE ICECS*, Dec 2012, pp. 400–403.

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

[14] F. J. Martin-Vega, F. Blanquez-Casado, F. J. López-Martínez, G. Gomez, and J. T. Entrambasaguas, "Further improvements in SOVA for high-throughput parallel turbo decoding," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 6–9, Jan 2015.

[15] Q. Huang, Q. Xiao, L. Quan, Z. Wang, and S. Wang, "Trimming soft-input soft-output Viterbi algorithms," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2952–2960, July 2016.

[16] G. Battail, "Pondération des symboles décodés par l'algorithme de Viterbi," *Ann. Telecommun.*, vol. 42, no. 1-2, pp. 31–38, Jan 1987.

[17] L. Lin and R. S. Cheng, "Improvements in SOVA-based decoding for turbocodes," in *Proc. IEEE Int. Conf. on Commun.*, vol. 3, June 1997, pp. 137–139.

[18] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

[19] A. J. Viterbi, "An intuitive justification and a simplified implementation of the map decoder for convolutional codes," *IEEE J. on Sel. Areas in Commun.*, vol. 16, no. 2, pp. 260–264, Feb 1998.

[20] F. Hemmati and D. Costello Jr, "Truncation error probability in Viterbi decoding," *IEEE Trans. Commun.*, vol. 25, pp. 530–532, 06 1977.

[21] M. P. C. Fossorier, F. Burkert, , and J. Hagenauer, "On the equivalence between SOVA and Max-Log-MAP decodings," *IEEE Commun. Lett.*, vol. 2, no. 5, pp. 137–139, May 1998.

[22] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Trans. Comm.*, vol. 51, no. 2, pp. 175–185, Feb 2003.

[23] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and optimization of an HSDPA turbo decoder ASIC," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 98–106, Jan 2009.