Non-Uniform Windowed Decoding Schedules for Spatially Coupled Codes

Ul Hassan, Najeeb; Pusane, Ali Emre; Lentmaier, Michael; Fettweis, Gerhard; Costello Jr., Daniel J.

Published in:
[Host publication title missing]

2013

Non-Uniform Windowed Decoding Schedules for Spatially Coupled Codes

Najeeb ul Hassan[†], Ali E. Pusane[*], Michael Lentmaier[+], Gerhard P. Fettweis[†], and Daniel J. Costello, Jr.[‡]
[†]Vodafone Chair Mobile Communications Systems, Dresden University of Technology (TU Dresden), Dresden, Germany, {najeeb.ul.hassan, fettweis}@ifn.et.tu-dresden.de
[*]Dept. of Electrical and Electronics Engineering, Bogazici University, Istanbul, Turkey, ali.pusane@boun.edu.tr
[+]Dept. of Electrical and Information Technology, Lund University, Lund, Sweden, michael.lentmaier@eit.lth.se
[‡]Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, Indiana, USA, costello.2@nd.edu

***Abstract*—Low-density parity-check convolutional (LDPCC) codes, also known as spatially coupled LDPC codes, can be decoded using a message passing algorithm. In order to limit decoding latency and complexity, windowed decoding can be applied. Updates within the window can be performed either in parallel or serially. However, simulation results show that uniform updating schedules do not provide the expected reduction in complexity when applied within the window. Hence we propose non-uniform schedules for updating the nodes based on measured improvements in the bit error rate. Nodes within the window that stop showing any improvement are excluded from the update list for the next iteration. This results in a reduction of up to 50% in complexity compared to uniform window schedules.**

I. Introduction

Spatially coupled (convolutional) LDPC codes are gaining more and more attention due to their ability to achieve the maximum a posteriori (MAP) threshold of an underlying LDPC block code with suboptimal message passing decoding. However, to achieve the MAP threshold [1] [2], a large termination length L must to be considered. This increases the complexity and the structural latency [3] of the decoder. The convolutional structure of the code allows us to define a sliding window decoder [4] of size W. As a result, decoding latency and decoding complexity become independent of L. Moreover, due to the fact that $W \ll L$, the storage requirements for the decoder are reduced by a factor of L/W compared to a non-windowed decoder.

LDPC convolutional (LDPCC) codes can be represented by a bipartite Tanner graph consisting of check and variable nodes. Considering an additive white Gaussian noise (AWGN) channel, messages in the form of log likelihood ratios (LLRs) are passed between the check and variable nodes iteratively. At each iteration, node updates can be performed according to a parallel (flooding) or serial (on-demand) rule. In both schedules, all the nodes in the graph are updated at every decoding iteration. Hence we refer to these schedules as the *uniform parallel* and *uniform serial* schedules, respectively. Some variants of the serial update rule to reduce the decoding complexity of the LDPC block codes have been discussed in [5].

In contrast to uniform schedules, a schedule can also be defined such that only a portion of the Tanner graph is updated in a decoding iteration. An example of a non-uniform schedule is forced convergence decoding [6], where the nodes that converge to a strong belief after the first few iterations are deactivated for the following iterations. Another example of a non-uniform schedule is a sliding window decoder, where the node updates are limited by the window size ($W < L$). Normally, the nodes within the window are updated according to a uniform schedule, where all the nodes within the decoding window are updated in every decoding iteration. Note that the node updates within the window can still be performed either in parallel or serially. However, it has been shown in [7] that uniform window schedules are not optimum with respect to decoding complexity.

For non-uniform window schedules, since only nodes in the first position are decoded before the window slides to the next position, a reduction in decoding complexity results by avoiding unnecessary updates to nodes not directly connected to the first position in the window. Only nodes that show improvement based on their bit error rate (BER) compared to the previous iteration are updated in the next iteration. In [8], the difference between the value of a message passed along an edge before and after an update is used to determine the update order for a uniform serial schedule. By contrast, the reduction in complexity due to non-uniform window schedules is mainly due to avoiding unnecessary updates, and we do not consider adapting the order of node updates based on the BER improvement. The analysis of non-uniform window schedules in [7] is limited to the binary erasure channel (BEC) using density evolution, which assumes infinite code length. In this paper, we evaluate the performance of non-uniform window schedules for finite length codes and introduce a periodic non-uniform window schedule to further decrease decoding complexity. We show that up to a 50% reduction in complexity is possible compared to a uniform window schedule. We further consider both parallel and serial update rules for the non-uniform window schedules.

This work was supported in part by the DFG in the CRC 912 HAEC, European Social Fund in the framework of the Young Investigators Group 3DCSI, TÜBİTAK Grant 111E276, EU FP7 Marie Curie IRG Grant 268264, NSF Grant CCF-CCF-1161754, and by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM# (Grant agreement no. 318306).

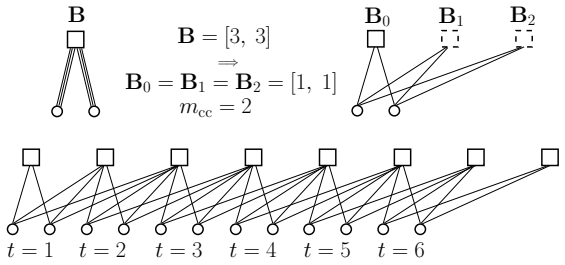


Fig. 1. Illustration of edge spreading: the protograph of a (3,6)-regular block code with base matrix $\mathbf{B}$ is repeated $L = 6$ times and the edges are spread over time according to the component base matrices $\mathbf{B}_0$, $\mathbf{B}_1$, and $\mathbf{B}_2$, resulting in a terminated LDPCC code.

II. Spatially Coupled LDPC Codes

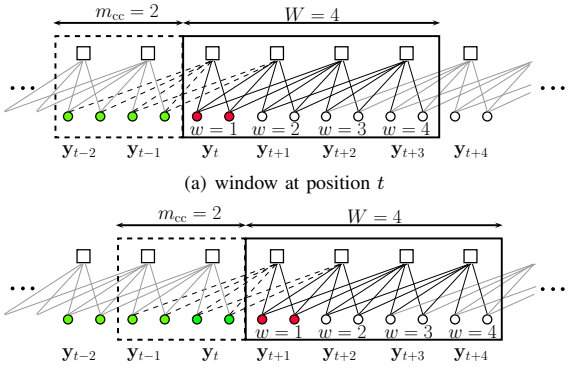
Consider the transmission of a sequence of codewords $\mathbf{v}_t$, $t = 1, \ldots, L$, using a protograph based LDPC code. A protograph is a small bipartite graph consisting of n_c check nodes and n_v variable nodes and is represented by its bi-adjacency matrix $\mathbf{B}$, called the *base matrix*. An essential feature of LDPCC codes [9] is that the blocks at different time instants are interconnected. Instead of encoding all codewords independently, the blocks $\mathbf{v}_t$ are *coupled* by the encoder to other time instants. The largest distance between a pair of coupled blocks defines the memory m_{cc} of the convolutional code. The coupling of consecutive blocks can be achieved by an *edge spreading* procedure [10] that distributes the edges from variable nodes at time t among equivalent check nodes at times $t+i$, $i = 0, \ldots, m_{\text{cc}}$. This procedure is illustrated in Fig. 1 for a (3,6)-regular protograph with $n_v = 2$ and $n_c = 1$ and base matrix $\mathbf{B} = [3, 3]$. In order to maintain the degree distribution and structure of the original protograph, a valid edge spreading should satisfy the condition

$$\sum_{i=0}^{m_{\text{cc}}} \mathbf{B}_i = \mathbf{B} \ . \tag{1}$$

The resulting code can be described by means of a terminated *convolutional protograph* with base matrix

$$\mathbf{B}_{[1,L]} = \begin{bmatrix} \mathbf{B}_0 & & \\ \vdots & \ddots & \\ \mathbf{B}_{m_{\text{cc}}} & & \mathbf{B}_0 \\ & \ddots & \vdots \\ & & \mathbf{B}_{m_{\text{cc}}} \end{bmatrix}_{(L+m_{\text{cc}})n_c \times Ln_v} . \tag{2}$$

The corresponding sequence of coupled code blocks forms a codeword $\mathbf{v} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t, \ldots, \mathbf{v}_L]$ of a terminated convolutional code. Note that the $m_{\text{cc}}n_c$ additional check nodes result in a rate loss due to termination. The block code with disconnected protographs corresponds to the special case $m_{\text{cc}} = 0$ and $\mathbf{B}_0 = \mathbf{B}$. The parity-check matrix $\mathbf{H}$ of a terminated LDPCC code can then be obtained by applying a graph lifting procedure [11] that replaces each 1 in $\mathbf{B}_{[1,L]}$ by an $N \times N$ permutation matrix and each 0 by an $N \times N$ all-zero matrix.



(a) window at position t

(b) window at position $t+1$

Fig. 2. Window decoder of size $W = 4$. The green variable nodes represent decoded nodes and the red nodes are the target nodes within the current window. The dashed lines represent the read access to the m_{cc} previously decoded blocks.

III. Decoding Schedules for Spatially Coupled LDPC Codes

A. Uniform Decoding Schedules

Standard uniform parallel or serial decoding schedules can be applied across the sequence of L coupled codewords. A uniform parallel (flooding) schedule simultaneously updates all the variable nodes followed by simultaneous check node updates. Alternatively, updates can be performed using a pre-defined uniform serial schedule. In this case, a check (variable) node is updated immediately after all its neighboring variable (check) nodes have been updated. Such a schedule ensures that the newly computed messages from the neighbors of a check (variable) node are used in the same decoding iteration. In the uniform parallel schedule, these newly computed messages can only be utilized in the next decoding iteration. As a result the required number of iterations for a uniform serial schedule reduces to almost half compared to a uniform parallel schedule [5]. An efficient implementation for both uniform parallel and uniform serial decoding schedules is a pipeline decoder [9] [12]. The pipeline decoder can decode the input sequence continuously but at the expense of a large delay. It has also been shown that the number of required decoding iterations for the uniform decoding schedules increases linearly with the parameter L [13]. In order to limit the decoding complexity and delay, a sliding window decoder was proposed in [4], which is a one-sided variant of the windowed decoder introduced in [2] for density evolution analysis.

B. Window Decoding Schedule

Consider two coupled code blocks $\mathbf{v}_t$ and $\mathbf{v}_{t'}$, where $t' \geq t + m_{\text{cc}} + 1$ and $t, t' \in [1\ L]$. Due to the memory m_{cc} of the convolutional code, these code blocks do not have any check nodes in common. This characteristic is exploited to define a latency constrained sliding window decoder. The size of the windowed decoder, W, must be at least $m_{\text{cc}} + 1$ code blocks, which is the largest distance between two coupled code blocks. Figure 2 shows an example of a sliding window decoder with $W = 4$. The window consists of W received

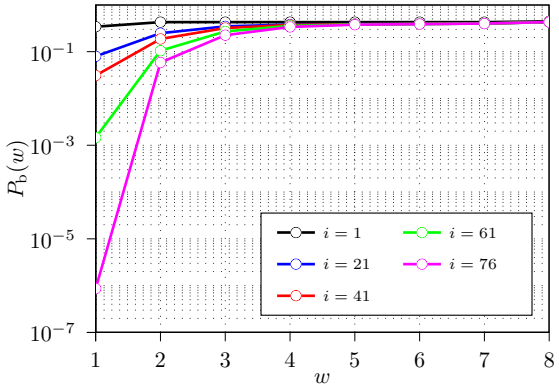


Fig. 3. Density evolution results for the BER $P_{\mathrm{b}}(w), w = 1, \ldots, W$, within the decoding window over a BEC with the uniform parallel window schedule as iterations i are performed ($W = 8$, $\epsilon = 0.48$).

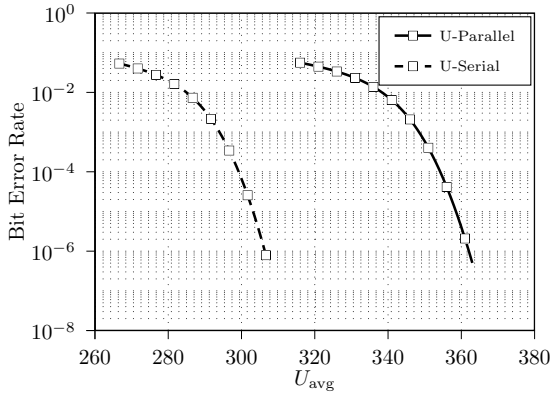


Fig. 4. Density evolution results for the BER as a function of the average number of node updates (U_{avg}) when uniform parallel (U-Parallel) and uniform serial (U-Serial) window schedules are applied, $W = 8$, $\epsilon = 0.48$.

blocks, $\mathbf{y}_{t+w-1}, w = 1, \ldots, W$. Additionally, at each window position, only the nodes in received block $\mathbf{y}_t$ are decoded (represented in red), and hence they are termed *target nodes*. After the received block $\mathbf{y}_t$ is decoded or some maximum number of iterations $I_{\max}$ is performed, the window slides to the next position as shown in Fig. 2(b). Moreover, due to the memory of the code, the sliding window decoder also requires read access to the m_{cc} previously decoded blocks (represented by the dashed lines), as shown in Fig. 2.

In sliding window decoding, nodes at position t are updated in multiple window positions. Hence we define the decoding complexity, the average number of node updates required to decode a symbol node, as [13]

$$U_{\mathrm{avg}} = \frac{1}{L} \sum_{t=1}^{L} U_t \ , \tag{3}$$

where U_t denotes the number of times a variable node at position t is updated during the iterative decoding process.

The edge spreading $\mathbf{B}_0 = [2,\ 2], \mathbf{B}_1 = [1,\ 1]$ containing multiple edges in $\mathbf{B}_0$ is now considered as an example. This type of edge spreading was proposed in [14] to achieve a certain desired BER with as small a W as possible. Note that the structural latency of the code is proportional to W. Hence a small value of W is desired for latency constrained applications [3].

Uniform Window Schedules: The updates within the decoding window are performed according to a uniform parallel (flooding) schedule, where all the nodes within a window are updated in parallel. Alternatively, all the nodes in the decoding window can be updated serially. Since all the nodes within the window are updated, we refer to these schedules as the *uniform parallel window* and *uniform serial window* schedules, respectively. Note that these schedules are uniform only with respect to the decoding window.

We analyze the convergence of the BER over a BEC with erasure probability of $\epsilon = 0.48$ when a uniform parallel window schedule is applied. The analysis is performed using density evolution. Figure 3 shows the BER $P_{\mathrm{b}}(w)$ for the nodes $w = 1, \ldots, W$ within the decoding window. It can be observed that the BER for the nodes other than the target nodes ($w > 1$) change little with iterations. The same behavior can be observed for the uniform serial window schedule (not shown here). Figure 4 shows the BER for the target nodes ($P_{\mathrm{b}}(w = 1)$) as a function of U_{avg} when uniform parallel and uniform serial window schedules are applied for a BEC with $\epsilon = 0.48$. We see that the uniform serial window schedule converges faster compared to the uniform parallel window schedule. But the gain in decoding convergence speed is much less than a factor of two[1]. In order to further reduce decoding complexity, the nodes not directly connected to the target nodes can be switched off after the first few decoding iterations are performed. This avoids unnecessary updates and results in a significant complexity reduction compared to uniform window schedules.

IV. Non-Uniform Window Schedules

A non-uniform improvement based window schedule was proposed in [7]. The variable nodes within the window that stop showing any improvement in their BER compared to the previous iteration are excluded from the update list for the following iterations along with the check nodes connected to these variable nodes. Hence it is not necessary that all the nodes within the window be updated in every decoding iteration. Such schedules are referred as *non-uniform window schedules*. Next we focus on non-uniform parallel window schedules. Non-uniform serial window schedules are discussed later in the section.

A. Non-Uniform Parallel Window Schedules

In order to define a non-uniform schedule based on the BER improvement, an estimate $\hat{P}_{\mathrm{b}}(w), w = 1, \ldots, W$ is required. These estimates are calculated using the output LLRs of the variable nodes at position w. This was termed a *soft BER* estimate in [3].

[1]A factor of two in complexity reduction is visible when a uniform serial schedule is applied to an LDPC block code or to an LDPCC code across the entire graph of length L [5] [12], rather than across a window of size W.

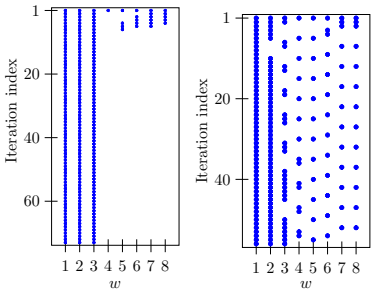


Fig. 5. Schedules adopted by a non-uniform improvement based window schedule for $\epsilon = 0.48$: (a) without using the P_{R} parameter and (b) with $P_{\mathrm{R}} = 5$.

Let $L_{\mathrm{ch}}(k)$ denote the input channel LLR for variable node k. The messages sent from variable node k to check node j and check node j to variable node k are denoted as L_{v_k,c_j} and L_{c_j,v_k}, respectively. Algorithm 1 defines a non-uniform parallel window schedule. The inputs are the window size W, the desired BER $P_{\mathrm{b}}^{\max}$ for the target nodes ($w = 1$), and the maximum number of iterations $I_{\max}$ within the window. In the initialization phase, the function `CalculateSoftBER(w)` computes the average soft BER estimates for the nodes within the window using output LLRs (L_{out}), where the output LLR $L_{\mathrm{out}}(k)$ for variable node k is computed as

$$L_{\mathrm{out}}(k) = L_{\mathrm{ch}}(k) + \sum_{l \in \mathcal{N}(v_k)} L_{c_l,v_k} \tag{4}$$

and $\mathcal{N}(v_k)$ is the set of check nodes connected to variable node k. Additionally the vector **`updateList[`**w**`]`** is initialized to *true* so that all the positions w within the window are updated in the first decoding iteration. The iterative process starts from line 4. The loops at lines 5 and 8 simultaneously update all the check and variable nodes at position w, respectively, for which the **`updateList[`**w**`]`** is *true*. The node updates are performed using the functions `UpdateCheckNodesAt(w)` and `UpdateVariableNodesAt(w)` for check and variable nodes given by (5) and (6), respectively,

$$L_{c_j,v_k} = 2\tanh^{-1}\left(\prod_{l \in \mathcal{N}(c_j)\backslash k} \tanh\left(\frac{L_{v_l,c_j}}{2}\right)\right), \tag{5}$$

$$L_{v_k,c_j} = L_{\mathrm{ch}}(k) + \sum_{l \in \mathcal{N}(v_k)\backslash j} L_{c_l,v_k}. \tag{6}$$

The function `CalculateOutputLLR(w)` calculates L_{out} for the nodes at position w. These are used to compute the new BER estimates $\hat{P}_{\mathrm{b,new}}$. The vector **`updateList[`**w**`]`** for the next iteration is recalculated based on the new ($\hat{P}_{\mathrm{b,new}}$) and old ($\hat{P}_{\mathrm{b,old}}$) soft BER estimates. The window positions w for which the BER improvement exceeds the parameter θ are updated in the next decoding iteration. The loop at line 4 is terminated if $\hat{P}_{\mathrm{b}}(w = 1)$ is less than $P_{\mathrm{b}}^{\max}$. Otherwise, the iterations continue until $I_{\max}$ is reached. Algorithm 1 gives the decoding algorithm for the first window position. For the following window positions, the soft BER in the initialization phase is calculated only for the new incoming nodes in the right side of the window.

Algorithm 1: Non-uniform parallel window schedule

Inputs: W, $P_{\mathrm{b}}^{\max}$, $I_{\max}$

```
   /* initialization phase                                   */
1  for w ← 1 to W do
2      P̂_b,old(w) ← CalculateSoftBER(w);
3      updateList [w] ← true;
   /* iterations start here                                  */
4  for i ← 1 to I_max do
5      for w ← 1 to W do
6          if updateList [w] then
7              UpdateCheckNodesAt(w);
8      for w ← 1 to W do
9          if updateList [w] then
10             UpdateVariableNodesAt(w);
11             CalculateOutputLLR(w);
12     for w ← 1 to W do
13         P̂_b,new(w) ← CalculateSoftBER(w);
14         if P̂_b,new(w) ≤ θ · P̂_b,old(w) then
15             updateList [w] ← true;
16             P̂_b,old(w) ← P̂_b,new(w);
17         else
18             updateList [w] ← false;
19     if P̂_b,new(1) ≤ P_b^max then
20         break;
```

Figure 5(a) shows the resultant schedule when Algorithm 1 is applied to the BEC with $W = 8$ and $\epsilon = 0.48$. The positions $w = 1, \ldots, W$ that are updated in an iteration i are represented by a blue circle. It is observed that positions $w = 4, 5, \ldots, 8$ stop showing any improvement and hence are not updated after the first few decoding iterations. This is consistent with the observation made in Fig. 3. Figure 6 shows the number of updates U_t when a non-uniform parallel window schedule (blue triangles) is applied to the BEC with $\theta = 0.99$[2]and $\epsilon = 0.48$. It is observed that the deactivation of nodes within the window results in a significant reduction in decoding complexity compared to the uniform parallel window schedule[3] (black squares).

In [7], the authors determined that a non-uniform window schedule can also be obtained by performing an exhaustive search of all possible decoding schedules, under some as-

[2]The value of θ that gives the lowest complexity was experimentally determined.

[3]In general, for the uniform schedules, the number of required iterations for the first window position is more than for the later positions because previously calculated messages are reused when the window slides. This can be seen by the jump in complexity for uniform schedules observed in Fig. 6 and Fig. 7 at $t = W$.

Algorithm 2: Additions to Algorithm 1

Additional input: P_{R}

3a counter $[w] \leftarrow 0$;

18a counter $[w] \leftarrow$ counter $[w] + 1$;
18b **if** counter $[w] == P_{\mathrm{R}}$ **then**
18c updateList $[w] \leftarrow$ *true*;
18d counter $[w] \leftarrow 0$;

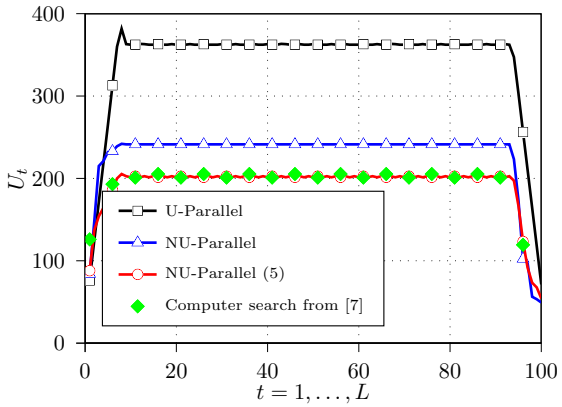


Fig. 6. Number of updates U_t for symbols at time $t = 1, \ldots, L$ for uniform and non-uniform parallel schedules. $W = 8, \epsilon = 0.48, L = 100$.

sumptions that help reduce the search space. However, the complexity of such a search quickly becomes untractable as the window size increases. Furthermore, the schedule must be computed in advance using density evolution. This makes it infeasible to be applied to the practical systems.

The schedules obtained from an exhaustive computer search were periodic, and Fig. 6 shows that the computer search schedule (green diamonds) in this case is less complex than the proposed improvement based non-uniform parallel window schedule. Hence, in order to incorporate this periodic feature in the proposed schedule, we force nodes to be updated that were not updated in some number P_{R} of previous iterations.

Algorithm 2 gives the additions required to Algorithm 1 to incorporate the periodicity factor P_{R}. Figure 5(b) shows the schedule adopted when $P_{\mathrm{R}} = 5$ is applied. Although the node updates per iteration increase due to periodically updating deactivated nodes, Fig. 6 suggests that overall complexity is reduced by 20%. Furthermore, the non-uniform parallel schedule with periodic updates achieves essentially the same performance as the computer search schedule.

B. Non-Uniform Serial Window Schedules

The node updates can also be performed using a non-uniform serial window schedule. Algorithm 3 gives the modifications required to the update rule in Algorithm 1. In the serial update, before updating the check nodes at position w (line 9 in Algorithm 3), all the neighboring variable nodes are asked to produce messages along the edges connected to check nodes at position w. A drawback of a serial decoding

Algorithm 3: Non-uniform serial window schedule

5 **for** $w \leftarrow 1$ **to** W **do**
6 **if** updateList $[w]$ **then**
7 **forall the** $v_i \in \mathcal{N}(c_w)$ **do**
8 Calculate and propagate L_{v_i,c_w} if updateList $[w']$ is $true$ where w' is the position of v_i within the window
9 `UpdateCheckNodesAt(w);`
10 **for** $w \leftarrow 1$ **to** W **do**
11 `CalculateOutputLLR(w);`

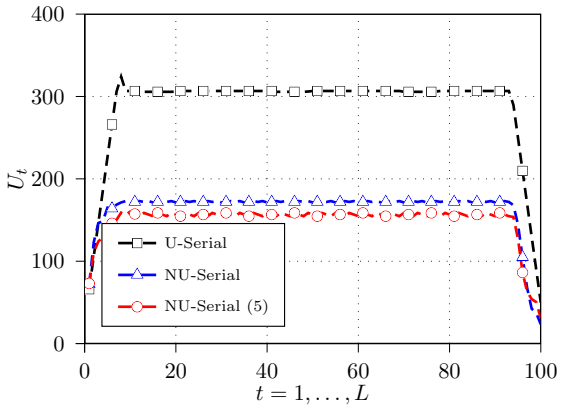


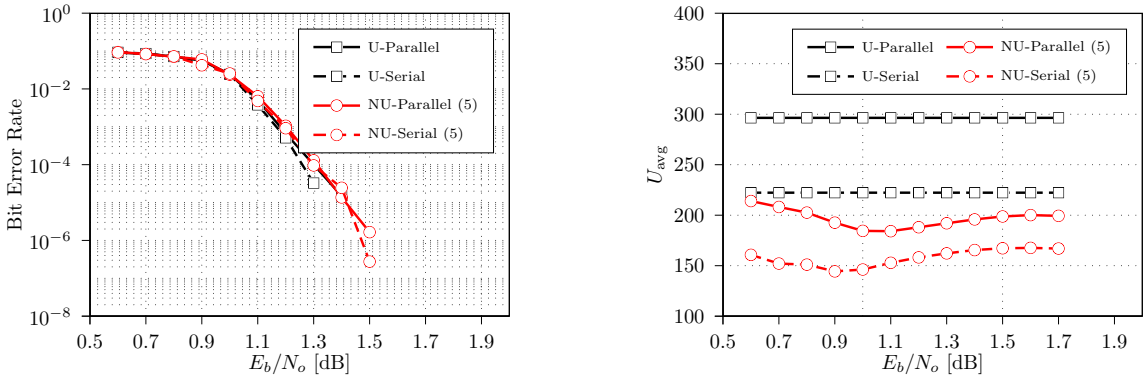
Fig. 7. Number of updates U_t for symbols at time $t = 1, \ldots, L$ for uniform and non-uniform serial schedules. $W = 8, \epsilon = 0.48, L = 100$.

schedule is that the iterations cannot be fully parallelized, i.e. all the variable and check nodes within the decoding window cannot be updated simultaneously. However, with the schedule of Algorithm 3, it is possible to update all the check nodes at position w simultaneously, thus allowing us to partially parallelize an iteration.

The parameter P_R can also be incorporated in the serial schedule using the additions from Algorithm 2. Figure 7 compares the values U_t for the uniform (black squares) and non-uniform (blue triangles) serial window schedules. Similar to the non-uniform parallel window schedule, including periodicity with $P_{\mathrm{R}} = 5$ in the non-uniform serial window schedule (red circles) also provides a reduction in complexity. Compared to Fig. 6, it is observed that the serial update rule provides a gain with respect to the parallel update rule. Furthermore, the non-uniform serial window schedule results in more than a 50% reduction in complexity compared to the uniform parallel window schedule.

V. Performance Evaluation for Finite Length Codes

In this section, the performance of a finite length LDPCC code is analyzed for an AWGN channel. The finite length code is generated using a graph lifting procedure with a lifting factor of $N = 500$, so that each coupled code block $\mathbf{v}_t$ is of length 1000 (Nn_v) code bits. We use a window of size



(a) Bit error rate performance

(b) Average node updates

Fig. 8. Simulation results for the BER and average number of node updates U_{avg} for an AWGN channel with $N = 500$, $W = 8$. For parallel window schedules $I_{\max} = 40$ is chosen and $I_{\max} = 30$ is chosen for serial window schedules.

$W = 8$ and apply the uniform and non-uniform window schedules described in the previous sections[4]. Furthermore, we only consider non-uniform window schedules with P_{R}, since they achieve the smallest complexity. The maximum number of iterations is selected such that the BER for the considered schedules is similar, and we do not apply any stopping criterion for iterations within a window[5]. Also, since serial schedules converge faster to the same BER performance compared to parallel schedules, we choose a slightly larger value of $I_{\max}$ for the parallel window schedules compared to the serial window schedules. Figure 8(a) shows BER curves for the uniform and non-uniform window schedules, and we see that the choice of $I_{\max}$ results in no significant loss in performance for non-uniform window schedules compared to uniform window schedules. Figure 8(b) shows U_{avg} for the considered schedules. We see that the non-uniform parallel window schedule provides a $35-40\%$ reduction in complexity compared to the uniform parallel window schedule, and the gain can be increased up to 50% by using a serial non-uniform window schedule.

VI. Conclusion

We analyzed the potential reduction in decoding complexity achieved when non-uniform schedules are applied to sliding window decoding of spatially coupled codes compared to uniform schedules. The nodes within the window are switched off once they stop showing BER improvement. This results in a significant reduction in complexity without any loss in performance. Moreover, we showed that periodically updating the deactivated nodes within the window results in a further reduction in decoding complexity and achieves the same complexity as the exhaustive computer search schedule from [7]. Finally, both parallel and serial update rules within the window were considered.

[4]The structural latency of the sliding window decoder in this case is $WNn_v = 8000$ code bits [3].

[5]This implies that the iteration stopping condition at line 19 of Algorithm 1 is not used and $I_{\max}$ iterations are performed for every window position.

References

[1] S. Kudekar, T. Richardson, and R. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 803–834, Feb. 2011.

[2] M. Lentmaier, A. Sridharan, D. Costello, and K. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5274 –5289, Oct. 2010.

[3] N. Ul Hassan, M. Lentmaier, and G. Fettweis, "Comparison of LDPC block and LDPC convolutional codes based on their decoding latency," in *Proc. 7'th International Symposium on Turbo Codes & Iterative Information Processing*, Aug. 2012.

[4] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, "Windowed decoding of protograph-based LDPC convolutional codes over erasure channels," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2303–2320, Apr. 2012.

[5] E. Sharon, N. Presman, and S. Litsyn, "Convergence analysis of generalized serial message-passing schedules," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 1013 –1024, Aug. 2009.

[6] E. Zimmermann, W. Rave, and G. Fettweis, "Forced convergence decoding of LDPC codes - EXIT chart analysis and combination with node complexity reduction techniques," *11th European Wireless Conference - Next Generation Wireless and Mobile Communications and Services (European Wireless)*, pp. 1 –8, Apr. 2005.

[7] N. Ul Hassan, A. E. Pusane, M. Lentmaier, G. P. Fettweis, and D. J. Costello Jr, "Reduced complexity window decoding schedules for coupled LDPC codes," in *Proc. IEEE Information Theory Workshop*, Sep. 2012, pp. 20 – 24.

[8] A. Vila Casado, M. Griot, and R. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *IEEE International Conference on Communications (ICC)*, Jun. 2007, pp. 932 –937.

[9] A. Jimenez Felstrom and K. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181 –2191, Sep. 1999.

[10] M. Lentmaier, G. Fettweis, K. Zigangirov, and D. Costello, "Approaching capacity with asymptotically regular LDPC codes," in *Proc. Information Theory and Applications Workshop (ITA)*, Feb. 2009, pp. 173 –177.

[11] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," *In IPN Progress Report 42-154, JPL*, Aug. 2003.

[12] A. Pusane, A. Feltstrom, A. Sridharan, M. Lentmaier, K. Zigangirov, and D. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 56, no. 7, pp. 1060 –1069, Jul. 2008.

[13] M. Lentmaier, M. Prenda, and G. Fettweis, "Efficient message passing scheduling for terminated LDPC convolutional codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Aug. 2011, pp. 1826 –1830.

[14] M. Papaleo, A. Iyengar, P. Siegel, J. Wolf, and G. Corazza, "Windowed erasure decoding of LDPC convolutional codes," in *IEEE Information Theory Workshop (ITW)*, Jan. 2010, pp. 1 –5.