Reduced Complexity Window Decoding Schedules for Coupled LDPC 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,
{michael.lentmaier, 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 Engineering, University of Notre Dame, Notre Dame, Indiana, USA, costello.2@nd.edu

***Abstract*—Window decoding schedules are very attractive for message passing decoding of spatially coupled LDPC codes. They take advantage of the inherent convolutional code structure and allow continuous transmission with low decoding latency and complexity. In this paper we show that the decoding complexity can be further reduced if suitable message passing schedules are applied within the decoding window. An improvement based schedule is presented that easily adapts to different ensemble structures, window sizes, and channel parameters. Its combination with a serial (on-demand) schedule is also considered. Results from a computer search based schedule are shown for comparison.**

I. Introduction

When a sequence of LDPC code blocks are *coupled* together to form a terminated convolutional code [1], a remarkable *threshold saturation* effect is observed: the belief propagation (BP) decoding threshold of the coupled ensembles converges to the optimal maximum a-posteriori probability (MAP) decoding threshold of the underlying block ensemble as the number L of coupled blocks increases [2], [3]. On the other hand, for channel parameters close to the threshold, the decoding complexity per symbol increases linearly with L if a standard BP block decoder is used [4]. Such a complexity increase occurs not only for a parallel (flooding) schedule but also for a serial (on-demand) schedule [5], [6], [7] if every check node and variable node is updated at each decoding iteration, i.e., if the schedule is *uniform*. Examples of non-uniform schedules for which the complexity is independent of L are the improvement based schedule (MPS-III) introduced in [8] and the sliding window decoder discussed in [9]. The window decoder has the additional advantage of a limited decoding delay.

While the node updates in [9] are restricted to a sliding window of size W, they are still performed according to a uniform parallel schedule *within* the decoding window. In this paper we investigate the combination of sliding window decoding with different serial and/or non-uniform update rules within the window, which are shown to further reduce the decoding complexity.

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, and NSF Grant CCF-1165714.

II. Efficient Message Passing Schedules

A. Parallel and Serial Schedules

For BP decoding of LDPC block codes the message passing schedule that is most commonly used is the *flooding schedule*: in each decoding iteration all check nodes are updated first and then all variable nodes are updated. At every node update all incoming messages are processed to create new outgoing messages that are then passed back to each neighboring node. Since the processing at the nodes can be performed independently in parallel, this schedule is also known as a *parallel schedule*.

It is a consequence of the parallel node update that newly calculated messages cannot be directly reused within the same decoding iteration. Such a direct update of messages, which leads to a faster message flow through the graph, is enabled by a *serial schedule*: in each decoding iteration all check nodes are serially updated in a predefined order. At each check node update, all neighboring variable nodes are requested to produce new messages for the active check node using their latest set of incoming messages. These newly produced messages are then used by the active check node to produce new outgoing messages to be sent back to each neighboring variable node. This *serial check node* (or *on-demand variable node*) schedule can also be modified to a dual *serial variable node* (or *on-demand check node*) schedule. Such direct message update schedules typically reduce the required number of iterations by a factor two [5], [6], [10] and further reductions are possible with some generalizations [7].

B. Spatial Coupling: LDPC Convolutional Codes

Consider the transmission of a sequence of codewords $\mathbf{v}_t$, $t = 1, \ldots, L$, using a protograph based LDPC code. An essential feature of LDPC convolutional (LDPCC) codes [1] 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 over various other time instants. The maximal 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 [11] that divides the edges from variable nodes at time t among equivalent check nodes at times $t + i$, $i = 0, \ldots, m_{\mathrm{cc}}$. This procedure is illustrated in Fig. 1 for a

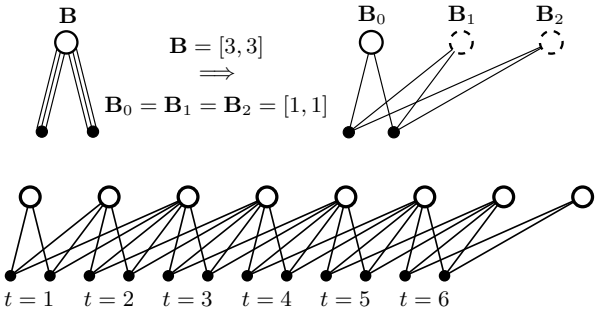

Fig. 1. Illustration of edge spreading: the protograph of a (3,6)-regular block code with base matrix $\mathbf{B}$ is repeated $L = 6$ times and the edges are spread over time according to the component base matrices $\mathbf{B}_0$, $\mathbf{B}_1$, and $\mathbf{B}_2$, resulting in a terminated LDPC convolutional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

(3,6)-regular protograph with $n_v = 2$ variable nodes, $n_c = 1$ check node, and base matrix $\mathbf{B} = [3, 3]$. In order to maintain the degree distribution and structure of the original ensemble, a valid edge spreading should satisfy the condition

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

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

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

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 LDPC convolutional code. Note that the $m_{\mathrm{cc}}n_c$ additional check nodes result in a rate loss due to termination. The block coding ensemble with disconnected protographs corresponds to the special case $m_{\mathrm{cc}} = 0$ and $\mathbf{B}_0 = \mathbf{B}$. The ensembles considered in this paper are defined in Table I.

C. Decoding Schedules for Coupled LDPC codes

It is widely known that the special structure of convolutional codes is well-suited for efficient pipeline decoding [1], [10]. Note that the output of the original pipeline decoder in [1] is equivalent to the output of a standard flooding schedule decoder applied to the overall sequence of length L. However, the pipeline decoder allows for continuous windowed transmission without any termination or, more practically, termination after an arbitrarily large number L of consecutively encoded blocks. A pipeline decoder with a serial (or on-demand) schedule was considered in [10].

A major drawback of the flooding schedule (and hence the classical pipeline decoder), is that the number of required decoding iterations increases with the parameter L for channel parameters close to the threshold of the coupled ensemble [4]. Such a dramatic increase in decoding complexity can be avoided by employing efficient message passing schedules

TABLE I
COMPONENT MATRICES USED IN THE EDGE SPREADING:

Code	$\mathbf{B}$	m_{cc}	$\mathbf{B}_i$
Ensemble A	[3 3]	2	$\mathbf{B}_{0,1,2} = [1\ 1]$
Ensemble B	[3 3]	1	$\mathbf{B}_0 = [2\ 2], \mathbf{B}_1 = [1\ 1]$
Ensemble C	[4 4]	2	$\mathbf{B}_0 = [2\ 2], \mathbf{B}_{1,2} = [1\ 1]$

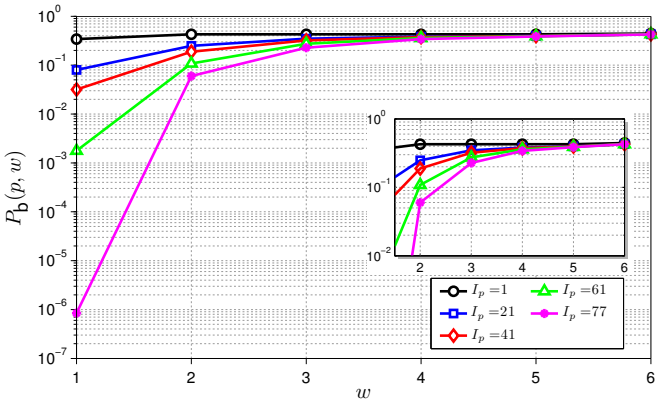

Fig. 2. Bit error probability $P_{\mathrm{b}}(p, w), w = 1, \ldots, W$, for a window at position p for Ensemble B on a BEC with the flooding schedule ($W = 6$, $\epsilon = 0.48$).

that make use of the convolutional structure of coupled LDPC codes. An attractive, practical implementation of such a schedule is the sliding window decoder proposed in [12], [9], for which both latency and complexity are independent of L. A two-sided version of such a window decoder has also been considered using a density evolution analysis in [3].

In the following, using density evolution, we investigate alternative message passing schedules *within* the decoding window that reduce the complexity compared to the windowed flooding schedules that have been considered so far.

III. Window Decoding Schedules with Further Complexity Reduction

The sliding window decoder of size W operates on a section of $W \cdot n_c$ rows and $W \cdot n_v$ columns of the protograph $\mathbf{B}_{[1,L]}$ in (2). For a window at position p, only the first n_v symbols, termed *target symbols*, are decoded. After a certain number of iterations I_p are performed at position p, the window slides n_c rows down and n_v columns right in $\mathbf{B}_{[1,L]}$.

The probabilities of error $P_{\mathrm{b}}(p, w), w = 1, \ldots, W$, within a window at position p for Ensemble B on a binary erasure channel (BEC) with erasure probability ϵ and the flooding schedule are shown in Fig. 2. As iterations are performed the bit error probability for the target symbols within the window ($w = 1$) converges to a target error probability $P_{\mathrm{b}}^{\max} = 10^{-6}$. It can be observed from the inset plot of Fig. 2 that increasing the number of iterations has little effect on the probabilities of the variable nodes for $w > 1$. We now discuss two schedules that utilize this property of the window decoder and apply a non-uniform update schedule within a window in order to reduce the complexity of the decoder.

A. Computer search based decoding schedules

One possible way to identify good schedules to decode a given code ensemble within a window is to perform an exhaustive search of all possible decoding schedules. In order to make this schedule search tractable some assumptions need to be made regarding the decoding schedule structure.

We start by defining a parameter ρ_W of a decoding schedule that describes the ratio of the number of node updates to the number required by the flooding schedule within a window of size W. As an example, for a window decoder with $W = 4$ (consisting of 4 check nodes and 8 variable nodes), a decoding schedule with $\rho_W = 1$ updates 4 check nodes and 8 variable nodes, which could include multiple activations of some nodes. Similarly, $\rho_W = 3$ corresponds to a total of 12 check node and 24 variable node updates. We further define an *iteration period* consisting of $\rho_W \cdot W$ iterations and require that each node within the window is updated at least once within this period.

Another assumption we make regarding the decoding schedule structure is that the variable nodes at a given time are activated following the activation of the associated check nodes. Let c_w and v_w, $w = 1, \ldots, W$, denote the n_c check nodes and the n_v variable nodes of the protograph, respectively, at each position w inside a decoding window of size W. We then apply the constraint that the variable nodes at position w are activated following the activation of the check nodes at position w. This shrinks the search space for the optimum decoding schedule drastically, since we just need to enumerate all possible activations of check nodes and the variable node activations are determined from these.

Following the general philosophy of the window decoder, one final assumption is made to enforce causality on the order of the check node activations, i.e., within a window a check node $c_{w'}$ cannot be updated after $c_{w''}$ for $1 \leq w' < w'' \leq W$.

Based on these assumptions a decoding schedule is optimized and repeatedly applied to the window until a maximum number of repetitions, or iteration periods, is reached or a correct decision is achieved. For Ensemble B with $W = 4$, one iteration period for an optimized schedule with $\rho_W = 2.5$, consisting of 10 iterations, is presented in Fig. 3(a), where the dots indicate the order in which the check nodes are updated within one iteration period. Throughout the iterations plotted in Fig. 3(a), a slight bias is observed towards activating the check nodes with lower indices (located towards the left side of the window) more frequently than the others. A similar and more pronounced effect is observed in Fig. 3(b), where a decoding schedule optimized for the same ensemble with window size $W = 8$ and $\rho_W = 2.25$ is plotted. Although these decoding schedules do not alter the iterative decoding threshold of the ensemble, they nevertheless reduce the number of iterations required to achieve the threshold and hence reduce the decoding complexity. As an example, at $\epsilon = 0.48564$, which is the threshold of the ensemble for $W = 8$, the optimized decoding schedule requires 964 iteration periods for reaching $P_{\mathrm{b}}^{\max} = 10^{-6}$. This corresponds to $964 \times 2.25 = 2169$ flooding iterations, whereas applying the classical flooding schedule would require 4636 decoding iterations.

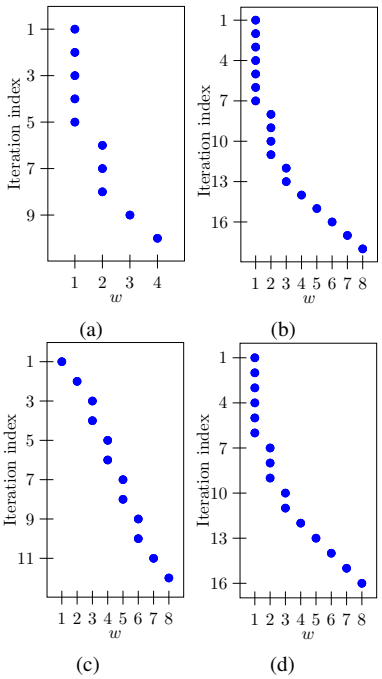

Fig. 3. One iteration period of the optimized decoding schedule for (a) Ensemble B, $W = 4$, $\rho_W = 2.5$ (b) Ensemble B, $W = 8$, $\rho_W = 2.25$ (c) Ensemble A, $W = 8$, $\rho_W = 1.5$ and (d) Ensemble C, $W = 8$, $\rho_W = 2$ ($\epsilon = 0.48$).

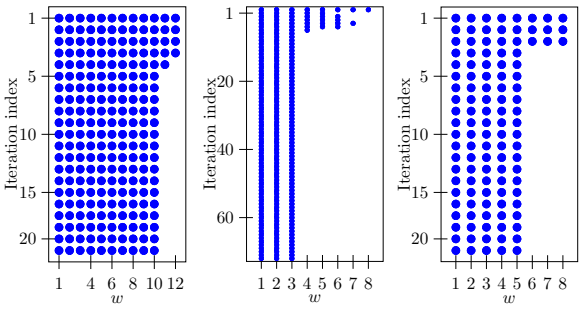

Fig. 4. Schedules adopted by improvement based schedule for (a) Ensemble A, $W = 12$ (b) Ensemble B, $W = 8$ (c) Ensemble C, $W = 8$ ($\epsilon = 0.48$).

A similar decoding schedule optimization for Ensemble A, whose result is given in Fig. 3(c), reveals that not all ensembles enjoy decoding schedules with more frequent check node activations on the left side of the window. The optimum decoding schedule consists of activating the check and variable nodes in the window homogeneously, i.e., the optimum decoding schedule resembles the flooding schedule with no particular emphasis given to any part of the window. As another example, the optimum decoding schedule for Ensemble C is also presented in Fig. 3(d).

B. Improvement based decoding schedules

In this section we re-introduce the improvement based schedule proposed as MPS-III in [8]. In the case of a non-window decoder, it was observed in [8, Fig. 2] that, at the center of the protograph, there is little influence from the strong check nodes at the start of the iterative process.

Similarly, when windowed decoding is applied (see Fig. 2), there is little improvement in terms of bit error probability for the nodes at the right side of the window after the first few decoding iterations.

We apply the improvement based schedule within a window with an extra constraint that updates are allowed only on the nodes within the current window. In order to measure the improvement, we define the fractional bit error improvement $\Delta P_{\mathrm{b}}^{i}(w)$[1] for the nodes within the window in the ith iteration as follows,

$$\Delta P_{\mathrm{b}}^{i}(w) = \frac{(P_{\mathrm{b}}^{i-1}(w) - P_{\mathrm{b}}^{i}(w))}{P_{\mathrm{b}}^{i-1}(w)}, \quad 1 \leq i \leq I_p$$

In the first iteration all the nodes in the window are updated and the probability is initialized by the channel values ($P_{\mathrm{b}}^{0}(w) = \epsilon$). In the following iterations, the new probabilities are calculated for each position w within the window. The variable nodes in iteration i for which $\Delta P_{\mathrm{b}}^{i}(w) < \theta$ are excluded from the update list for the $(i+1)$th iteration along with the check nodes connected to those variable nodes. The iterations stop once the target error probability $P_{\mathrm{b}}^{\max}$ is reached for the target symbols.

By applying an improvement based schedule to a window decoder, the unnecessary updates on the right side of the window can be avoided. Figure 4 shows the nodes which are updated at a particular iteration for Ensembles A, B, and C for an erasure probability of $\epsilon = 0.48$. All iterations until the bit error probability converges to $P_{\mathrm{b}}^{\max}$ are shown in this figure since, unlike the schedules in Section III-A, this schedule is no longer periodic. In the first few iterations all the nodes show an improvement, and hence are updated, but after a certain number of decoding iterations are performed the fractional bit error improvement for the nodes on the right side of the window is less than θ and hence these nodes are not updated in the following iteration. This results in a non-uniform schedule and the decoding complexity is reduced by eliminating unnecessary updates within the window. We note that, for a given ensemble, the improvement based decoding schedule tends to resemble the computer search based schedule in the sense that a bias in terms of updates is observed for the nodes on the left side of the window.

IV. Results and Discussions

In this section different window decoding schedules are compared based on their decoding complexity. To fairly compare complexity, the following observations are made: 1) due to the sliding window nature of decoding, each node at position t is updated in multiple window positions and 2) in both schedules defined in the previous section, not all the nodes within the window are updated in each decoding iteration. Based on these observations, let U_t denote the number of times a variable node and check node at position t are updated during the iterative process. Furthermore, the average number of node updates required per symbol is defined as follows [8],

[1]The position p of the window is omitted here for simplicity of notation.

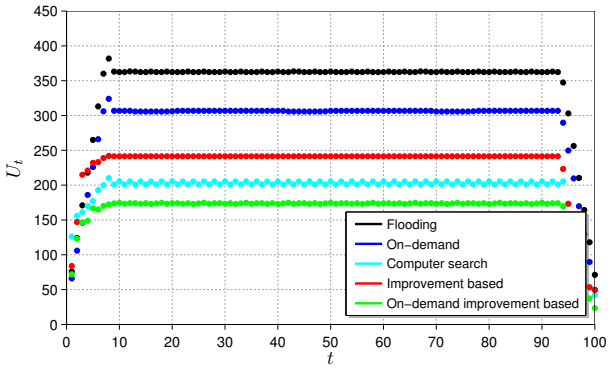

Fig. 5. Number of updates U_t for symbols at time $t = 1, \ldots, L$. Ensemble B, $W = 8$, $\epsilon = 0.48$, $L = 100$.

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

For a uniform schedule, all the nodes in the window are updated in every decoding iteration, and hence U_t is given as

$$U_t^{\mathrm{uniform}} = \sum_{p=t-W:1\leq p\leq L}^{t} I_p, \quad 1 \leq t \leq L \ . \tag{4}$$

The values U_t for the schedules discussed in the previous section are depicted in Fig. 5 for Ensemble B, window size $W = 8$, termination length $L = 100$, and erasure probability $\epsilon = 0.48$. For the improvement based schedule, the value of θ that gives the lowest complexity was experimentally determined to be 0.001 for all cases. A uniform on-demand schedule gives us an improvement in complexity, but one can even gain more by using the improvement based schedule together with on-demand, as shown in Fig. 5. Furthermore, if we restrict ourselves to parallel schedules, both the computer search based schedule and the improvement based schedule give a significant reduction in complexity compared to the uniform flooding schedule, due to avoiding unnecessary updates for nodes on the right side of the window.

The average updates required per symbol U_{avg}, as a function of the erasure probability, is shown in Fig. 6 for Ensembles B and C. The computer search based schedule provides some gain compared to the improvement based schedule, but it must be optimized for the channel value, whereas the improvement based schedule is more robust and can adapt to the channel conditions. The computer search based schedule presented in Fig. 6 is optimized for $\epsilon = 0.4856$ and $\epsilon = 0.4946$, respectively, and hence for $\epsilon < 0.44$ it is slightly more complex than the flooding schedule, whereas the improvement based schedule remains less complex than flooding for the full range of ϵ.

Figure 7 shows U_{avg} as a function of the window size W for Ensembles B and C. The average node updates for Ensemble C are lower than for Ensemble B for all schedules, due to the better threshold of the $(4, 8)$ LDPCC code compared to the $(3, 6)$ LDPCC code. The improvement based schedule gives a reduction in complexity for both ensembles compared to

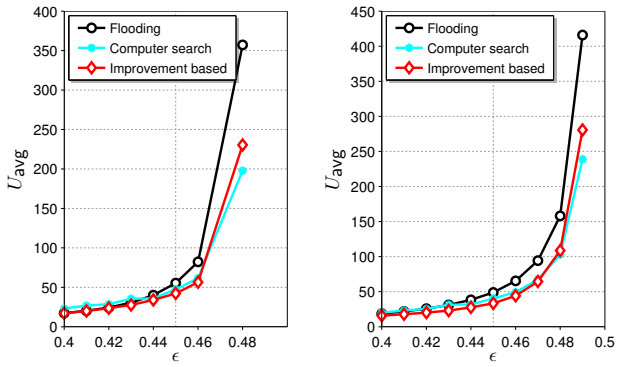

Fig. 6. Average node updates as a function of erasure probability for $W = 8$ (a) Ensemble B (b) Ensemble C.

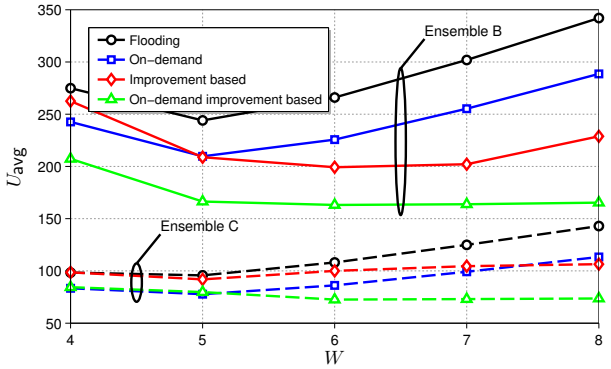

Fig. 7. Average node updates for Ensemble B (solid lines) and Ensemble C (dashed lines) as a function of W, $L = 100$, $\epsilon = 0.48$.

the uniform (flooding and on-demand) schedules. The gains are further enhanced by applying on-demand updates together with an improvement based schedule.

Figure 8 shows the complexity as a function of W for Ensemble A. Here we present the results for $W \geq 10$, since for $\epsilon = 0.48$ a larger window must be used for coupled ensembles with single edges in the convolutional protograph (see Fig. 1). The uniform on-demand schedule already provides a large improvement compared to the flooding schedule, and little additional gain is obtained by using the improvement based schedule. The same can also be observed for the $(4, 8)$-regular LDPCC code with single edges (not plotted here). This shows that for ensembles with single edges, the on-demand schedule is already close to optimal and little gain is obtained by using a non-uniform schedule. On the other hand, a large value of W is required for these ensembles, which, for the case of latency constrained applications, can be a problem [13]. The ensembles with double edges were proposed in [12] to achieve the target error probability with a smaller W. For this case the gain in terms of complexity when using the on-demand schedule is smaller than for the non-uniform schedules discussed here.

V. Conclusion

While previous considerations of window decoding assume a uniform flooding schedule within the decoding window, we have shown that, especially for channel parameters close

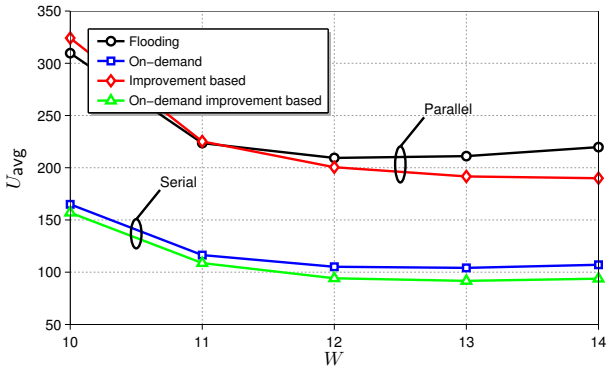

Fig. 8. Average node updates for Ensemble A as a function of W, $L = 100$, $\epsilon = 0.48$.

to the threshold, the complexity can be further reduced by schedules with non-uniform updates. Both parallel and serial (on-demand) update rules have been considered. A flexible adaptive improvement based schedule is proposed as an alternative to extensive computer searches for optimal schedules that become unfeasible as the window size increases and the channel parameter varies.

References

[1] 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.

[2] 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.

[3] 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.

[4] M. Lentmaier and G. Fettweis, "Coupled LDPC codes: Complexity aspects of threshold saturation," in *Proc. IEEE Information Theory Workshop (ITW)*, Oct. 2011, pp. 668 –672.

[5] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Proc. Thirty-Sixth Asilomar Conf. on Signals, Systems and Computers*, vol. 1, Nov. 2002, pp. 8–15.

[6] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A: Statistical Mechanics and its Applications*, vol. 330, pp. 259 – 270, 2003.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.