LDPC Decoders with Informed Dynamic Scheduling

Andres I. Vila Casado, Miguel Griot, and Richard D. Wesel, *Senior Member, IEEE*

***Abstract*—Low-Density Parity-Check (LDPC) codes are usually decoded by running an iterative belief-propagation (BP), or message-passing, algorithm over the factor graph of the code. The traditional message-passing scheduling, called flooding, consists of updating all the variable nodes in the graph, using the same pre-update information, followed by updating all the check nodes of the graph, again, using the same pre-update information. Recently, several studies show that sequential scheduling, in which messages are generated using the latest available information, significantly improves the convergence speed in terms of number of iterations. Sequential scheduling introduces the problem of finding the best sequence of message updates. We propose Informed Dynamic Scheduling (IDS) strategies that select the message-passing schedule according to the observed rate of change of the messages. In general, IDS strategies require computation to select the message to update but converge in fewer message updates because they focus on the part of the graph that has not converged. Moreover, IDS yields a lower error-rate performance than either flooding or sequential scheduling because IDS strategies overcome traditional trapping-set errors. This paper presents IDS strategies that address several issues including performance for short-blocklength codes, complexity, and implementability.**

***Index Terms*—Belief propagation, message-passing schedule, error-control codes, low-density parity-check codes.**

I. Introduction

BELIEF Propagation (BP) provides Maximum-Likelihood (ML) decoding over a cycle-free factor-graph representation of a code as shown in [1] and [2]. In some cases, BP over loopy factor graphs of channel codes has been shown to have near-ML performance. BP performs well on the (loopy) bi-partite factor graphs composed of variable nodes and check nodes that describe LDPC codes.

Loopy BP is an iterative algorithm and therefore requires a message-passing schedule. Flooding, or simultaneous scheduling, is the most popular scheduling strategy. In every iteration, flooding simultaneously updates all the variable nodes (with each update using the same set of pre-update data) and then, updates all the check nodes (again, with each update using the same pre-update information).

Paper approved by K. Narayanan, the Editor for Coding and Communication Theory of the IEEE Communications Society. Manuscript received June 22, 2007; revised July 8, 2008 and January 9, 2009.

This work was supported by the state of California and ST Microelectronics through UC discovery grant COM 03-10142. This paper was presented in part at the IEEE ICC Conference, Glasgow, Scotland, 2007.

A. I. Vila Casado is with Mojix Inc, Los Angeles, CA 90025 USA (e-mail: andres@mojix.com).

M. Griot is with QUALCOMM Inc, San Diego, CA 92121 USA (email: mgriot@ee.ucla.edu).

R. Wesel is with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: wesel@ee.ucla.edu).

Digital Object Identifier 10.1109/TCOMM.2010.09.070303

Recently, several papers have investigated different types of sequential, or non-simultaneous, scheduling strategies in BP LDPC decoding. With sequential scheduling, the messages are generated sequentially using the latest available information. Sequential scheduling was introduced as a sequence of check-node updates in [3] and [4] and as a sequence of variable-node updates in [5] and [6]. It is also presented in [7] under the name of Layered BP (LBP), in [8] and [9] as serial schedule, in [10] as row message passing, column message passing and row-column message passing, among others.

Monte-Carlo simulations and theoretical analysis in [3]-[10] show that sequential scheduling converges twice as fast as flooding when used in LDPC decoding. It has also been shown that sequential updating doesn't increase the decoding complexity per iteration, thus allowing the convergence speed increase at no cost [10], [11]. Although the literature listed above presents a variety of sequential schedules, they have very similar performance results [10]. In this paper, the sequential-scheduling strategy used for comparison is LBP, a sequence of check-node updates, as presented in [4] and [7].

Sequential updating introduces the problem of finding the ordering of message updates that results in the best convergence speed and/or code performance. The current state of the messages in the graph can be used to dynamically update the schedule, producing what we call Informed Dynamic Scheduling (IDS). We presented IDS in [12] and first published it in [13]. To our knowledge, these and the simultaneous work in [14] are the first works on the subject of dynamic scheduling for LDPC decoding. The author of [14] orders the updates in each iteration of a min-sum decoder using the concept of neighborhood reliabilities. We focus on a BP decoder and algorithms that select a specific message or group of messages to update based on metrics that indicate the importance of the messages. In other words, our algorithms update some messages multiple times before other messages are updated once, but in [14] every message is updated in every iteration.

Prior to these works, the only published well-defined IDS strategy is the Residual Belief Propagation (RBP) algorithm proposed in [15], which inspired our investigation. RBP was introduced for general sequential message passing, not specifically for LDPC decoding. RBP is a greedy algorithm that organizes the message updates according to the magnitude of the difference between the message generated in the current update and the message generated in the previous update. The intuition is that the larger this difference, the further this part of the graph is from convergence. Therefore, propagating messages with large differences first will make BP converge with fewer message updates.

Simulations show that RBP LDPC decoding has a better error-rate performance than LBP after a small number of message updates, but its error-rate performance for a large number of message updates is worse. This behavior is commonly found in greedy algorithms, which tend to arrive at a solution faster, but arrive at the correct solution less often. We propose a less-greedy IDS strategy in which all the outgoing messages of a check-node are generated simultaneously. We call this strategy Node-wise Scheduling (NS), and it delivers better frame and bit error rate than LBP across all possible numbers of message updates.

Both RBP and NS require precomputation of potential message updates in order to pick which message to update. This means that numerous messages are provisionally computed but not actually used in a message update which increases the complexity of the decoding. We propose using the min-sum check-node update, introduced in [16] and explained in [17], to simplify the computation of these provisional messages and significantly decrease the complexity of both strategies while maintaining the same performance. In our implementation, once a message is selected for an update, the provisional min-sum update value is replaced by the more precise sum-product value.

We study the behavior of IDS for different types of codes and applications such as short-blocklength LDPC codes, parallel decoders and lower-complexity decoders. We propose appropriate IDS strategies that work well for these applications.

This paper is organized as follows. Section II reviews LDPC decoding and the flooding and LBP schedules. Section III introduces RBP and NS for LDPC decoding as well as the min-sum IDS strategies. It also explains their performance by giving intuitive explanations and analyzing their behavior in the presence of traditional trapping-sets. Section IV analyzes the behavior of IDS on short-blocklength LDPC codes and introduces strategies that improve performance in this scenario. Section V introduces schedules more suitable for hardware implementation including a lower-complexity IDS strategy and a parallel IDS strategy. Section VI delivers the conclusions. Simulation results of the various message-passing schedules are presented along the way.

II. BP DECODING FOR LDPC CODES

In general, BP consists of the exchange of messages between the nodes of a graph. Each node generates and propagates messages to its neighbors based on its current incoming messages.

The LDPC code graph is a bi-partite graph composed by N variable nodes v_j for $j \in \{1,\dots,N\}$ that represent the codeword bits and M check nodes c_i for $i \in \{1,\dots,M\}$ that represent the parity-check equations. The exchanged messages correspond to the Log-Likelihood Ratio (LLR) of the probabilities of the bits. The sign of the LLR indicates the most likely value of the bit and the absolute value of the LLR gives the reliability of the message. Let the channel information LLR of the variable node v_j be denoted by C_{v_j}. Then, for any c_i and v_j that are connected, the two message generating functions are:

$$m_{v_j \to c_i} = \sum_{c_a \in \mathcal{N}(v_j) \backslash c_i} m_{c_a \to v_j} + C_{v_j}, \tag{1}$$

$$m_{c_i \to v_j} = 2 \times \operatorname{atanh}\left(\prod_{v_b \in \mathcal{N}(c_i) \backslash v_j} \tanh\left(\frac{m_{v_b \to c_i}}{2}\right)\right), \tag{2}$$

where $m_{v_j \to c_i}$ denotes the variable-to-check message from v_j to c_i and $\mathcal{N}(v_j) \backslash c_i$ denotes the neighbors of v_j excluding c_i.

BP decoding consists of the iterative update of the messages until a stopping rule is satisfied. In flooding scheduling, an iteration consists of the simultaneous update of all the messages $m_{v \to c}$ according to Eq. (1) followed by the simultaneous update of all the messages $m_{c \to v}$ according to Eq. (2). In sequential scheduling, an iteration consists of the sequential update of all the messages $m_{v \to c}$ as well as all the messages $m_{c \to v}$ in a specific pre-defined order. This pre-defined order is usually designed to allow the parallel processing of the messages. As an example, a possible LBP schedule is described in Algorithm 1. The algorithm stops if the decoded bits satisfy all the parity-check equations or a maximum number of iterations is reached.

Algorithm 1 LBP decoding for LDPC codes

```
Initialize all m_{c→v} = 0
for every i ∈ {1,...,M} do
    for every v_k ∈ N(c_i) do
        Generate and propagate m_{v_k→c_i}
    end for
    for every v_k ∈ N(c_i) do
        Generate and propagate m_{c_i→v_k}
    end for
end for
if Stopping rule is not satisfied then
    Position=2;
end if
```

III. INFORMED DYNAMIC SCHEDULING (IDS)

A. Residual Belief Propagation (RBP)

RBP, as introduced in [15], is an IDS strategy that updates messages according to an ordering metric called the residual. The message with the largest residual is updated first. A residual is the norm (defined over the message space) of the difference between the values of a message before and after an update. For a message $m_{n_i \to n_j}$ that goes from node n_i to node n_j, the residual is defined as:

$$r\left(m_{n_i \to n_j}\right) = \left\| m^{new}_{n_i \to n_j} - m^{old}_{n_i \to n_j} \right\|, \tag{3}$$

where the superscript new denotes the message to be propagated now and old denotes the message that was propagated the last time $m_{n_i \to n_j}$ was updated.

The intuitive justification of this approach is that as loopy BP converges, the differences between the messages before and after an update diminish. Therefore, if a message has a large residual, it means that it's located in a part of the graph

that hasn't converged yet. Therefore, propagating that message first should speed up convergence.

In LLR BP decoding, all the message spaces are one-dimensional (i.e. on the real line). Hence, the residuals are the absolute values of the difference of the LLRs.

Let us analyze the behavior of RBP decoding for LDPC codes in order to simplify the decoding algorithm. Initially, all the messages $m_{v\to c}$ are set to the value of their corresponding channel message C_v. The residuals of all the variable-to-check messages $r(m_{v\to c})$ are initially equal to 0. Then, without loss of generality, we assume that the message $m_{c_i\to v_j}$ has residual r^*, which is the largest of the graph. After $m_{c_i\to v_j}$ is propagated, only residuals $r(m_{v_j\to c_a})$ change, with $c_a \in \mathcal{N}(v_j)\backslash c_i$.

The new residuals $r(m_{v_j\to c_a})$ are equal to r^* because r^* was the change in the message $m_{c_i\to v_j}$ and Eq. (1) shows that the message-update operation of a variable node is the sum of incoming messages. Therefore, the just-updated messages $m_{v_j\to c_a}$ now have the largest residuals in the graph.

Assuming that propagating the messages $m_{v_j\to c_a}$ won't generate any new residuals larger than r^*, RBP can be simplified. Every time a message $m_{c\to v}$ is propagated, the outgoing messages from the variable node v will be updated and propagated. After propagation of the messages from the variable node v, all residuals for messages from variable nodes are again zero. This facilitates the scheduling since we need only to search for the largest $r(m_{c\to v})$ in order to find out the next message to be propagated. RBP LDPC decoding is formally described in Algorithm 2. Another way to implement RBP, presented in [15], is to create a priority queue of messages, ordered by the value of their residuals, so at each step the first message in the queue is updated and then the queue is reordered using the new information.

Algorithm 2 RBP decoding for LDPC codes

```
Initialize all m_{c→v} = 0
Initialize all m_{v_j→c_i} = C_j
Compute all r(m_{c→v})
Find r(m_{c_i→v_j}) = max r(m_{c→v})
Generate and propagate m_{c_i→v_j}
Set r(m_{c_i→v_j}) = 0
for every c_a ∈ N(v_j)\c_i do
    Generate and propagate m_{v_j→c_a}
    for every v_b ∈ N(c_a)\v_j do
        Compute r(m_{c_a→v_b})
    end for
end for
if Stopping rule is not satisfied then
    Position=4;
end if
```

There is an intuitive way to see how RBP decoding works for LDPC codes. Let us assume that at a certain moment in the decoding, there is a check node c_i with residuals $r(m_{c_i\to v_b}) = 0$ for every $v_b \in \mathcal{N}(c_i)$. Now let us assume that there is a change in the value of the message $m_{v_j\to c_i}$. The largest change in a check-to-variable message out of c_i (therefore the largest residual) will occur in the edge that corresponds to the incoming variable-to-check message with the lowest reliability (excluding the message $m_{v_j\to c_i}$). Let us denote by v_k the variable node that is the destination of the message that has the largest residual $r(m_{c_i\to v_k})$. Then, the message $m_{v_k\to c_i}$ has the smallest reliability out of all messages $m_{v_b\to c_i}$, with $v_b \in \mathcal{N}(c_i)\backslash v_j$.

This implies that, for this particular scenario, once there's a change in a variable-to-check message, RBP will propagate first the message to the variable node with the lowest reliability. This makes sense intuitively because the lowest-reliability variable node is more likely to be in error than the higher-reliability ones.

The negative effects of the greediness of RBP are apparent in the case of an unsatisfied check node connected to only one variable node in error. RBP will propagate first the message that will "correct" the variable node with the lowest reliability. Again, this is the most likely variable node to be in error. However, if that variable node was actually correct, the variable node in error will not be corrected and there will be one more error. The information from this new error will likely be propagated next because the largest changes in incoming messages tend to induce the largest residuals. This analysis helps us see why RBP corrects the most likely errors with fewer message updates, but has trouble correcting "difficult" errors as will be seen in the performance plots. We define "difficult" errors as the errors that need a large number of message updates to be corrected.

B. Node-wise Scheduling decoding for LDPC codes

In order to obtain a better performance, a less greedy scheduling strategy can be used. As noted earlier, some of the greediness of RBP comes from the fact that it tends to propagate first the message to the least reliable variable node. We propose to update and propagate simultaneously all the check-to-variable messages that correspond to the same check node c_i, instead of only updating and propagating the message with the largest residual $r(m_{c_i\to v_j})$. It can be seen, using the analysis presented earlier, that this algorithm is less likely to propagate the information from new errors in the next update. This is because there are many variable nodes that change as opposed to RBP where only one variable node changes. We call this less-greedy strategy Node-wise Scheduling (NS).

NS is similar to LBP; it is a sequence of check-node updates. However, unlike LBP, which follows a pre-determined order, the check node to be updated next is chosen dynamically according to a metric α_c. Each check node c_i has a metric α_{c_i} that corresponds to the largest $r(m_{c_i\to v_b})$ for every $v_b \in \mathcal{N}(c_i)$. NS is formally described in Algorithm 3.

NS converges in fewer message updates and yields a lower error-rate for a large number of message updates than LBP. NS does not converge quite as rapidly as RBP, but it does converge to the right answer far more often.

C. Min-sum IDS strategies

Both RBP and NS are more complex than traditional scheduling strategies because they incur two extra processes: residual computation and either a search for the largest residual, or a continuous ordering of the residuals. The residual computation requires the value of the message that would be

Algorithm 3 NS decoding for LDPC codes

Initialize all $m_{c\to v}=0$
Initialize all $m_{v_j\to c_i}=C_j$
Compute all α_c
Find $i=\arg\max_{u=\{1\ldots M\}}\alpha_{c_u}$
for every $v_k\in\mathcal{N}(c_i)$ **do**
  Generate and propagate $m_{c_i\to v_k}$
  Set $\alpha_{c_i}=0$
  for every $c_a\in\mathcal{N}(v_k)\backslash c_i$ **do**
    Generate and propagate $m_{v_k\to c_a}$
    Compute α_{c_a}
  end for
end for
if Stopping rule is not satisfied **then**
  Position=4;
end if

propagated. This requires additional complexity since there will be numerous message computations that will only be used to calculate residuals. We propose to use the lower-complexity min-sum check-node update approximation, [16] and [17], to compute the approximate residuals as follows,

$$\widetilde{r}_{n_i\to n_j}(m_k)=\left\|\widetilde{m}^{new}_{n_i\to n_j}-\widetilde{m}^{old}_{n_i\to n_j}\right\|, \tag{4}$$

where the tilde indicates the min-sum approximation. The min-sum update of all check-to-variable messages emanating from a particular check node begins by identifying the two variable-to-check messages with the two smallest reliabilities. The smallest reliability is assigned as the check-to-variable message reliability for all the edges except the edge that corresponds to the smallest variable-to-check reliability. The second smallest reliability is assigned to that remaining edge. The correct sign is computed for all the check-to-variable messages. Using the min-sum approximate residual function Eq. (4) in Algorithms 2 and 3, defines Approximate RBP (ARBP) decoding and Approximate NS (ANS) decoding respectively. These significantly less complex algorithms perform as well as the ones presented in Section III. Note that we only propose to use min-sum for the residual computation. For the actual propagation of messages we use the full update equations (1) and (2).

We compare the performance of traditional scheduling strategies and IDS strategies vs. the number of check-to-variable messages propagated. This comparison gives a clear idea of how BP decoding performance changes with different scheduling strategies. In traditional schedules like flooding and LBP, the algorithm checks if the decoded bits satisfy the parity-check equations after each iteration. In order to maintain the same stopping rule check complexity, our IDS algorithms check the stopping rule after the number of check-to-variable messages passed is a multiple of the total number of edges in the LDPC graph.

This approach compares the algorithms for the same number of messages passed (and thus, same message-generation complexity). However, for the same number of message updates, an IDS strategy has more overall complexity than traditional schedules because of the computation and sorting of residuals required for selecting the next update to be performed.

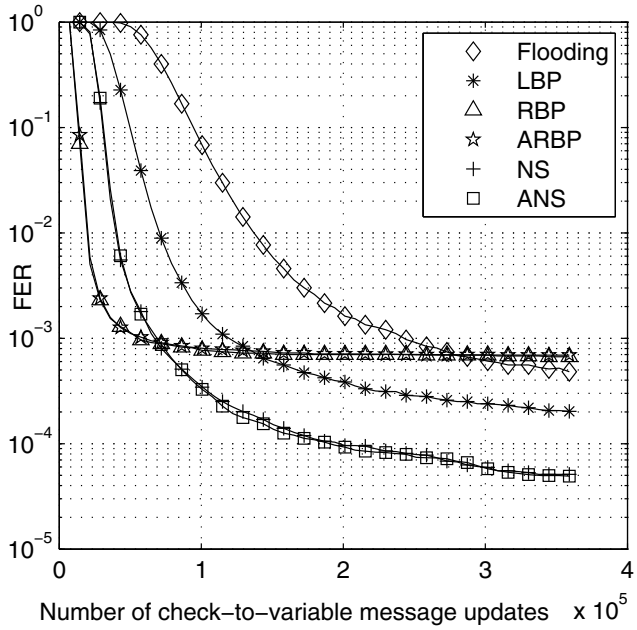

Fig. 1. FER vs. number of check-to-variable message updates of a blocklength-1944 rate-1/2 code using flooding, LBP, RBP, ARBP, NS and ANS for a fixed $E_b/N_o=1.75$ dB.

Fig. 1 shows the AWGN performance of the scheduling strategies discussed above, flooding, LBP, RBP, ARBP, NS, and ANS, vs. the number of check-to-variable message updates. The code simulated is a rate-1/2 LDPC code with blocklength 1944. The figure shows that RBP has a significantly better performance than LBP for a small number of message updates, but a sub-par performance for a larger number of message updates. This suggests that RBP has trouble with "difficult" errors as discussed earlier.

NS decoding, while not as good as RBP for a small number of message updates, performs much better than RBP after a number of message updates that would be typical in many common applications. NS also shows consistently better performance than LBP for any number of message updates. The results for flooding are shown for comparison purposes, and replicate the theoretical and empirical results of [4]-[10] that claim that flooding needs twice the number of iterations as sequential scheduling strategies such as LBP.

Fig. 1 also shows the performance of the approximate residual algorithms and compares them with the algorithms that use the exact residuals. It can be seen that both ARBP and ANS perform almost indistinguishably from RBP and NS respectively. We reiterate that the approximate residual diminishes the complexity of residual computation significantly, thus, making ARBP and ANS more attractive than their exact counterparts.

Fig. 2 and Fig. 3 show the performance of flooding, LBP, ARBP, and ANS for the blocklength 1944 rate-1/2 and rate 5/6 LDPC codes selected for the IEEE 802.11n standard [18]. These simulations were run for a high number of message updates (the equivalent of 200 LBP iterations) and show that

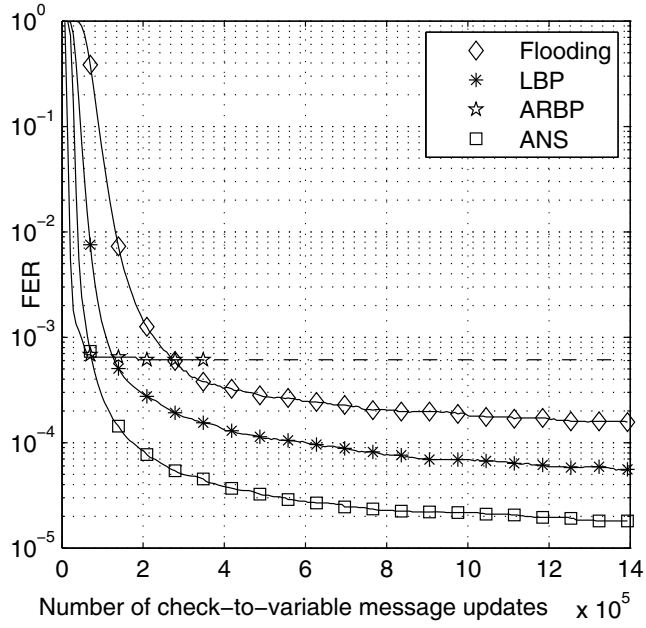

Fig. 2. FER vs. number of check-to-variable message updates of the 802.11n blocklength-1944 rate-1/2 code using flooding, LBP, ARBP and ANS for a fixed $E_b/N_o = 1.75$ dB. The dashed line extrapolates the performance of ARBP.

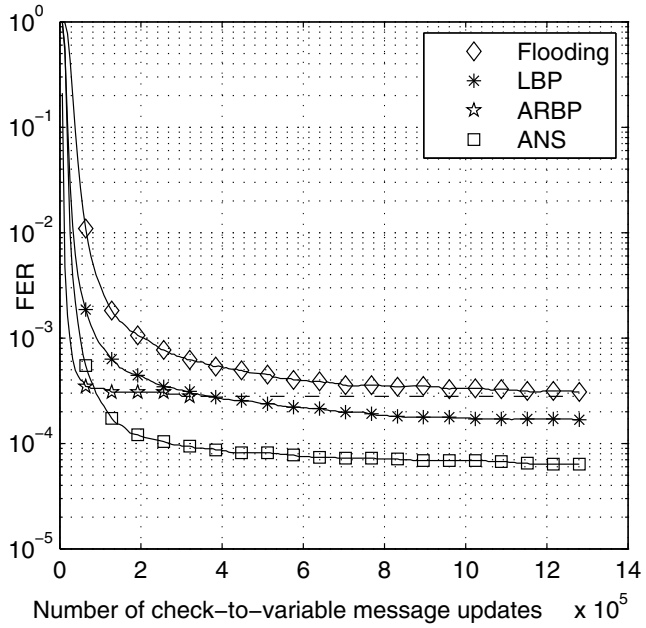

Fig. 3. FER vs. number of check-to-variable message updates of the 802.11n blocklength-1944 rate-5/6 802.11n code using flooding, LBP, ARBP and ANS for a fixed $SNR = 6.0$ dB. The dashed line extrapolates the performance of ARBP.

ANS still achieves a better FER performance than LBP. Fig. 3 also shows that even for high-rate codes, ANS converges in fewer message updates, and to a lower error-rate than LBP.

In the rest of the paper we focus on studying the behavior of ANS under different conditions. We focus on ANS decoding because both ANS and NS achieve very similar performance and ANS is less complex.

D. NS and ANS can overcome Trapping-set errors

Using residuals as the ordering metric for IDS was proposed to allow convergence in fewer message updates than LBP, the updates concentrate on the part of the graph that has not converged and thus it accelerates the convergence process. However, the previously presented performance results of both NS and ANS show that the algorithms also perform better than LBP for a large number of message updates. Fig. 2 and Fig. 3 show that for a large number of message updates (the equivalent of 200 LBP iterations), ANS reaches an FER that neither flooding nor LBP can reach even after many iterations.

Simulation results show that the lower error rates are achieved because NS and ANS allow the LDPC decoder to overcome many trapping sets. Trapping sets, or near-codewords, as defined in [19] and [20], are small variable-node sets such that the induced sub-graph has a small number of odd-degree neighbors. In [20], Richardson also mentions that the most troublesome trapping set errors are those where the odd-degree neighbors have degree 1 (in the induced sub-graph), and the even-degree neighbors have degree 2 (in the induced sub-graph).

Fig. 4 shows an example of how NS overcomes trapping sets. Updating the check node with the largest metric allows the decoding algorithm to focus on a part of the graph that hasn't converged yet. Thus, it is likely that NS solves the variable nodes in error by sequentially updating the degree-1 check nodes connected to them. If a variable node in a trapping set is corrected, the induced sub-graph of the variable-nodes-in-error will change as follows. At least one check node that was degree-2 becomes degree-1 (in the induced sub-graph of variable-node errors) after the variable-node correction. This check node is likely to be picked as the next check node to be updated by ANS because its messages will have large residuals produced when the check-to-variable messages changed signs. This update will probably correct another variable node in the trapping set.

ANS is more effective than LBP when solving trapping sets because it can sequentially update the degree-1 (in the induced sub-graph) check nodes that solve the errors. Since LBP is forced to update check nodes in a specific order, it is likely that degree-2 (in the induced sub-graph) check nodes, are updated first, which reinforces the errors.

We corroborated this analysis by studying the behavior of the decoders for the noise realizations that could not be solved by the LBP decoder of the 802.11n rate-1/2 code even after 200 iterations and that ANS solved in a very small number of message updates (less than the equivalent of 10 LBP iterations). We found that a large majority of the LBP errors in these cases are caused by trapping sets that ANS solved in the manner mentioned above. Also, Fig. 5 shows the performance of the blocklength-2640 Margulis code, proposed in [21], using flooding, LBP and ANS. The errors of the blocklength-2640 Margulis code at high SNRs have been shown to be dominated by trapping-set errors in [19] and [20]. The ANS performance improvement with respect to both flooding and LBP confirms that ANS can correct trapping-set errors that traditional scheduling strategies cannot.

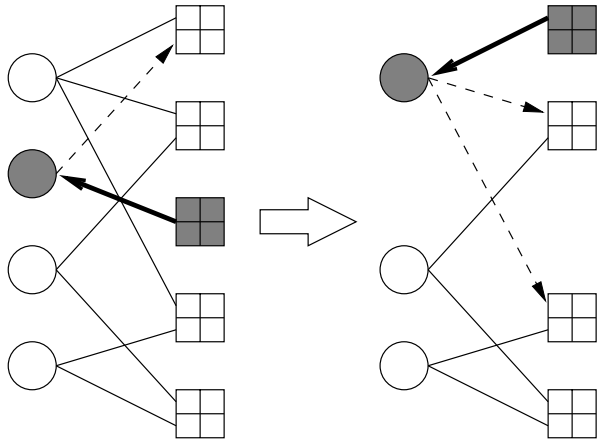

Fig. 4. Check-node update sequence that solves a trapping set. Shaded nodes represent the check node that is updated and the variable node that is corrected.

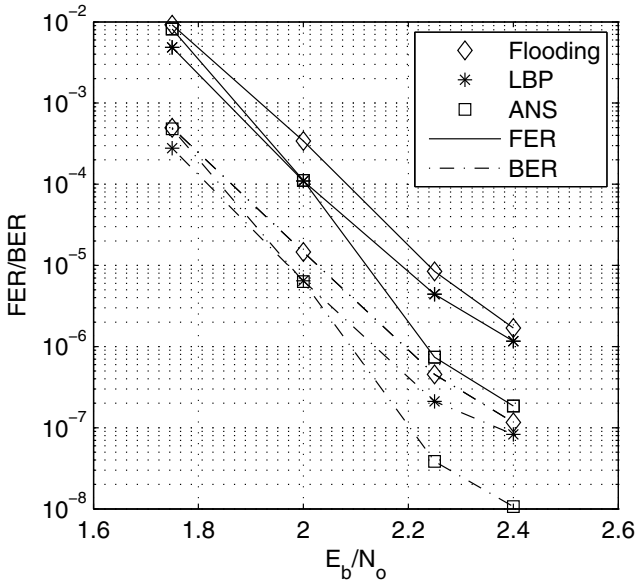

Fig. 5. AWGN performance of the blocklength-2640 Margulis code decoded by 3 different scheduling strategies: flooding, LBP and ANS. A maximum of the number of message updates in 50 LBP iterations was used.

IV. ANS FOR SHORT-BLOCKLENGTH LDPC CODES

A. Shortcomings of ANS decoding

ANS decoding, while better than traditional scheduling because it solves trapping sets, presents other types of errors that don't occur with LBP and flooding. They can be categorized into two classes: non-ML undetected errors and myopic errors.

We define non-ML undetected errors as undetected errors where the squared Euclidian distance between the decoded codeword and the received signal is larger than the squared Euclidian distance between the transmitted codeword and the received signal. This means that an ML decoder wouldn't make this mistake. Given its greedy nature, ANS makes more non-ML undetected errors than traditional LDPC decoders.

If there is a received signal that is near the border between two decoding regions (Voronoi regions), the initial BP message updates can take the decoder in any direction. ANS is more likely to make non-ML undetected errors than flooding or LBP because it has the ability to update only a part of the graph. This locally optimum approach is more likely to go in the wrong direction than the more global approach of LBP and flooding. The probability that ANS makes a non-ML undetected error decreases as the received signal is farther from the border. Thus, the negative effect of this behavior is more noticeable in the decoding of short-blocklength LDPC codes. Short-blocklength codes have a minimum Hamming distance small enough that the probability of receiving a signal near the border of two decoding regions is comparable to the probability of loopy-BP errors in high-SNR regimes.

Myopic decoding errors (the second type of errors cause by ANS) occur when the decoder focuses on a small number of check nodes despite the fact that many other bits in error need to be solved in a different part of the graph. Myopic errors become significant when the graph has many length-4 cycles. If ANS updates one of the check nodes in a length-4 cycle sub-graph, it is likely that the next check node to be chosen is the other one in the cycle because it receives *two* updated messages from the updated variable nodes. Thus, if the code has graph structures that contain many length-4 cycles, ANS can become stuck repeatedly updating the same small number of check nodes even if there are errors on other parts of the code. Simulations show that myopic errors are only significant for codes that present densely connected subgraphs such as randomly constructed short-blocklength codes that allow length-4 cycles.

B. IDS strategies for short-blocklength LDPC codes

We propose mixed strategies that combine LBP and ANS message updates in order to correct trapping-set errors and avoid the greedy ANS errors. The decoder starts by performing LBP iterations and then switches to ANS. Fixed LBP/ANS (F-LBP/ANS) first does a pre-determined number of LBP iterations ξ and then switches to ANS.

Given that one of the main advantages of ANS is the fact that it solves trapping sets, we propose another mixed strategy that we call Adaptive LBP/ANS (A-LBP/ANS). In A-LBP/ANS the decoder switches from LBP to ANS when the number of unsatisfied check nodes is below a certain value ζ. This makes sense given that the dominant trapping sets are those that have a small number of unsatisfied check nodes [20]. Thus, LBP will decode until it hits a trapping set with a small number of unsatisfied check nodes where ANS, better equipped to solve trapping sets, takes over. Since an ANS is more complex than LBP, these lower-complexity mixed strategies are also attractive for larger-blocklength codes because of their error-rate performance, which is similar to ANS. The optimal values of ξ and ζ can be found through Monte-Carlo simulations.

Table I shows the FER and Undetected FER (UFER) of 5 different rate-1/2 LDPC codes decoded using 5 different scheduling strategies. All the codes have blocklength 648, have the same variable-node degree distribution and the same concentrated check-node degree distribution. The UFER is defined as the total number of frames with undetected errors divided

TABLE I
FER AND UFER OF 5 DIFFERENT LDPC CODES DECODED BY 5 DIFFERENT SCHEDULING STRATEGIES: FLOODING, LBP, ANS, F-LBP/ANS WITH $\xi = 35$ AND A-LBP/ANS WITH $\zeta = 5$. THE CHANNEL USED IS AWGN WITH $E_b/N_o = 3\ dB$.

	Flooding		LBP		ANS		F-LBP/ANS		A-LBP/ANS	
Code	FER	UFER	FER	UFER	FER	UFER	FER	UFER	FER	UFER
A	4.2e-5	1.1e-6	1.7e-5	1.0e-6	5.8e-5	3.5e-6	3.9e-6	1.3e-6	3.9e-6	2.0e-6
B	1.6e-4	3.2e-6	1.1e-4	2.2e-6	3.4e-5	2.9e-5	2.0e-5	4.7e-6	1.5e-5	1.0e-5
C	3.4e-5	1.1e-6	1.6e-5	1.1e-6	5.1e-6	4.6e-6	3.0e-6	1.2e-6	2.6e-6	1.6e-6
D	4.4e-5	4.4e-6	3.0e-5	5.3e-6	1.9e-5	1.8e-5	1.2e-5	7.5e-6	1.1e-5	9.2e-6
E	2.2e-5	8.9e-7	6.5e-6	2.0e-6	5.8e-6	5.3e-6	3.3e-6	2.4e-6	4.2e-6	3.4e-6

by the total number of frames simulated. The simulations correspond to an AWGN channel with $E_b/N_o = 3$ dB and a maximum number of message updates equivalent to 50 LBP iterations was used.

Code A is a random code constructed using the Approximate Cycle EMD (ACE) and the Stopping-Set Check (SSC) graph constraint algorithms proposed in [22] and [23] respectively. These algorithms were designed to avoid the presence of small stopping sets. However, this code allows the presence of length-4 cycles. Code B was constructed to avoid only length-4 cycles. The ACE and the SSC algorithms were used to construct code C and length-4 cycles were avoided. Code D was constructed using the PEG algorithm presented in and [24]. The PEG algorithm is designed to locally maximize the girth of the graph as the matrix generation process goes on. This code has a girth of 6 so it doesn't have any length-4 cycles either. Finally, code E is an LDPC code selected for the IEEE 802.11n standard [18].

Let us analyze the performance of the traditional scheduling strategies: flooding and LBP. We corroborated experimentally that the detected errors, which are the difference between their FER and UFER values, are mostly trapping-set errors. Also, as expected, LBP performs better than flooding.

ANS outperforms LBP for all the codes except for code A. This is the only code in the group that has length-4 cycles and we experimentally corroborated that myopic errors described in Section IV dominate the performance of ANS decoding of this code at this SNR. As further proof, code C was designed to keep the same ACE and SSC graph constraints as code A while avoiding length-4 cycles. Code C doesn't incur in any ANS myopic errors. This demonstrates empirically that myopic errors dominate the error performance when the graph has several length-4 cycles. Furthermore, we see that the ANS UFERs are larger than their corresponding UFERs for flooding and LBP. This is due to an increase in the number of non-ML undetected errors as explained in Section IV. Table I shows that the ANS FER for the last four codes is dominated by the undetected errors; the percentages of frame errors that are undetected by ANS for codes B, C, D, and E are 85%, 90%, 95%, and 91% respectively.

Table I also shows the results of the mixed scheduling strategies. The fourth column shows the FER and UFER of F-LBP/ANS with $\xi = 35$. Hence, the decoder starts by performing 35 LBP iterations and switches to ANS. The fifth column shows the FER and UFER of A-LBP/ANS with $\zeta = 5$. Hence, the decoder starts by performing LBP iterations until the number of unsatisfied check nodes is less than or equal to 5. The values of ξ and ζ were not optimized. Both mixed strategies correct the ANS myopic errors of code A and also have a lower UFER than ANS in all the codes.

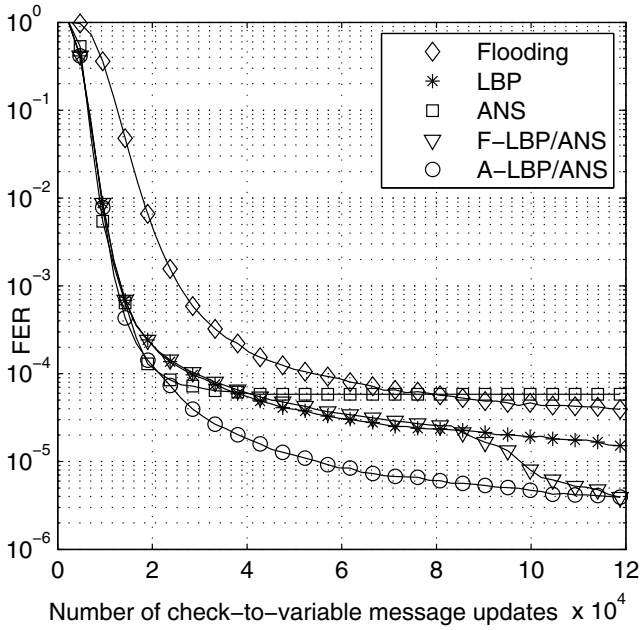

Fig. 6. AWGN Performance of code A vs. number of check-to-variable message updates for a fixed $E_b/N_o = 3$ dB. Results of 5 different scheduling strategies are presented: flooding, LBP, ANS, F-LBP/ANS with $\xi = 35$ and A-LBP/ANS with $\zeta = 5$.

Fig. 6 shows the FER of code A vs. the number of message updates. For a small number of message updates, ANS performs well. However, it presents an error floor at 6×10^{-5}. As mentioned before, a careful analysis of these errors showed that they were myopic errors due to the large number of length-4 cycles. No ANS myopic errors were observed for codes that don't have length-4 cycles. Furthermore, Fig. 6 shows that both mixed strategies perform very well when compared to LBP and flooding.

Fig. 7 shows the FER and UFER of code C for a maximum number of message updates equivalent to 50 LBP iterations. The FER of the three IDS strategies closely approach their respective UFER for a high SNR. Also, while ANS presents a larger UFER than LBP and flooding at 3 dB, the mixed strategies' UFERs are similar to those of LBP and flooding. This shows that the mixed strategies provide a good combination of harvesting the trapping-set correction capability of ANS while avoiding the errors generated by ANS's greediness. Mixed strategies are also less computationally demanding than pure ANS.

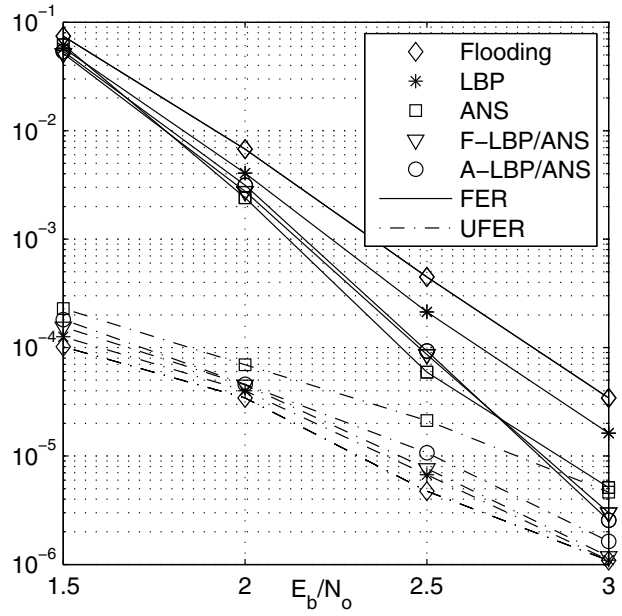

Fig. 7. AWGN FER and UFER vs E_b/N_o for for code C decoded by 5 different scheduling strategies: flooding, LBP, ANS, F-LBP/ANS with $\xi = 35$ and A-LBP/ANS with $\zeta = 5$.

V. Implementation strategies for IDS

A. Lower-Complexity ANS (LC-ANS)

As mentioned in Section III, ANS selects the check node to be updated based on a metric α_c, which is the largest approximate residual of the check-to-variable messages that are generated in the check node. Thus, in order to generate α_{c_i} we must compute the approximate residuals of all the check-to-variable messages of check node c_i and find the largest one.

In order to reduce these computations we propose to infer which edges are more likely to have the larger residuals of each check node based on the following considerations. The largest α_{c_i} metric corresponds to the largest residual of all the check-to-variable messages in the graph. It is likely that the largest residual in the graph corresponds to a check-to-variable message that has a different sign before and after the update. It is also likely that among the check-to-variable messages that change their sign after the update, the largest residual corresponds to the message that has the largest reliability after the update.

Lower-Complexity ANS (LC-ANS) selects the check node to be updated based on a simplified check-node metric α_c^{LC} that focuses on the messages with the largest reliability after the update. Within the same check node, the check-to-variable messages with larger reliability correspond to the edges that have the variable-to-check messages with the smaller reliability. We define α_c^{LC} as the sum of the two residuals that correspond to the edges that have the two variable-to-check messages with the smallest reliability. Given that we use min-sum to compute the residuals, the two variable-to-check messages with the smallest reliability are known. Thus, in order to generate $\alpha_{c_i}^{LC}$, only two residuals are computed and then summed which is significantly less complex than generating α_{c_i}.

B. Parallel Decoding

The possibility of having several processors computing messages at the same time during the LDPC decoding has become an intense area of research and an important reason why LDPC codes are so successful. Furthermore, codes with a specific structure have been shown to allow LBP decoding while maintaining the same parallelism degree obtained for flooding decoding [4]. In principle, the idea of having an ordered sequence of updates that uses the most recent information as much as possible isn't compatible with the idea of simultaneously computing messages. However, it is likely that one of the incoming variable-to-check messages to the check-nodes with the largest residuals changes in the previous update. Hence, it is possible that several parallel processors can work on different parts of the graph that haven't converged yet while still using the most recent information.

We propose Parallel-ANS (P-ANS) as an IDS strategy that is very similar to ANS where instead of updating only one check-node, the one with the largest α_{c_i} metric, the p nodes that have the largest α_{c_i} metrics are updated simultaneously. These p check nodes are not designed to work in parallel, unlike the p check-nodes of a $p \times p$ sub-matrix as defined in [4].

However, parallel processing may be implemented extending the hardware solutions presented in [25]. For instance, if one or more check-nodes have one or more variable nodes in common, they will all use the same previous information and compute the incremental variations that are afterwards combined in the variable-node update. There are hardware issues, such as memory clashes, that still need to be carefully addressed when implementing P-ANS.

Fig. 8 shows the FER and BER of another rate-1/2 blocklength-1944 LDPC code decoded using 6 different scheduling strategies: flooding, LBP, ANS, P-ANS with $p = 81$, LC-ANS and A-LBP/ANS with $\zeta = 5$. The code was designed to have no length-4 cycles and the maximum number of message updates was set to the equivalent of 50 LBP iterations. Both LC-ANS and A-LBP/ANS perform closely to ANS while requiring a lower complexity. Fig. 8 also shows that P-ANS performs very close to ANS across all SNRs.

Similar conclusions can be obtained from the error rate results of several IDS strategies for decoding the IEEE 802.11n rate-1/2 blocklength-1944 LDPC presented on Fig. 9. Figs. 8 and 9 also show that the performance improvement of IDS strategies increases as the SNR increases. This is explained by the fact that as the SNR increases, trapping-set errors become dominant. This suggests that IDS strategies can significantly improve the error-floor of LDPC codes. Finally, Figs. 8 and 9 show that the performance improvement of IDS strategies are larger in the FER curves than in the BER curves. This is explained by the fact that the advantages of IDS strategies reside mostly on the fact that they can correct trapping set errors. Since trapping set errors induce a frame error with a small number bit errors, overcoming trapping sets can reduce the FER significantly while providing a smaller reduction in BER (which is dominated by frames that fail with many errors).

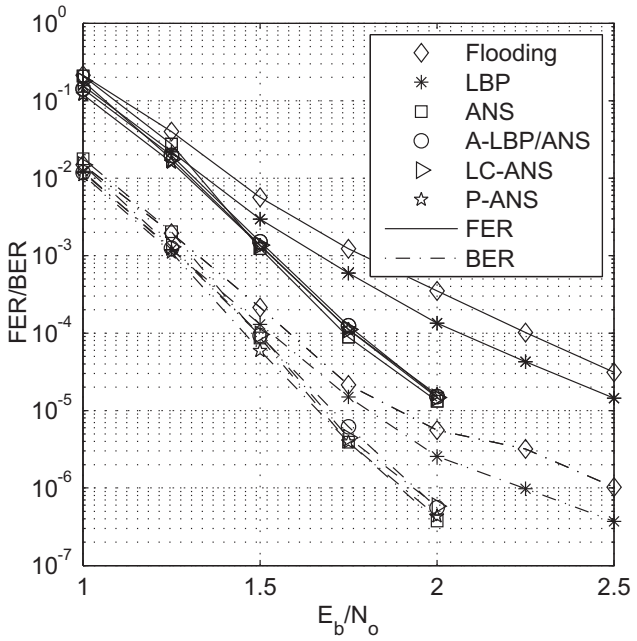

Fig. 8. AWGN performance of a rate-1/2 blocklength-1944 LDPC code decoded by 6 different scheduling strategies: flooding, LBP, ANS, A-LBP/ANS with $\zeta = 5$, LC-ANS and P-ANS.

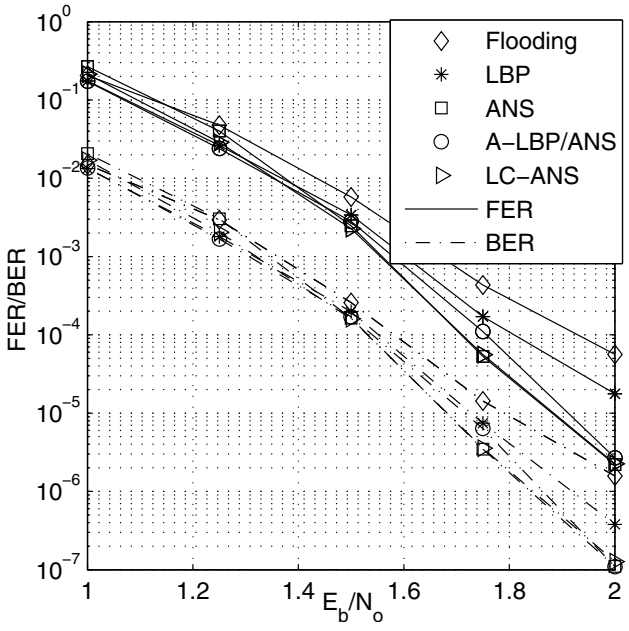

Fig. 9. AWGN performance of the IEEE 802.11n rate-1/2 blocklength-1944 LDPC code decoded by 5 different scheduling strategies: flooding, LBP, ANS, A-LBP/ANS with $\zeta = 5$ and LC-ANS.

VI. Conclusions

While maintaining the same message-generation functions as well as maintaining or reducing the total number of messages propagated in the graph, IDS can improve the performance of BP LDPC decoding at the cost of an increase in complexity to select the messages to be propagated.

Except for poorly constructed codes with many length four cycles, NS and its simplification ANS perform better than LBP for any target error-rate and any number of message updates. Both strategies achieve lower error-rates by overcoming trapping-set errors that LBP cannot solve.

However, for short-blocklength codes there is an increase in the number of non-ML undetected errors that significantly affect the performance of ANS in high-SNR regimes. Also, for codes that have a large number of length-4 cycles ANS makes myopic errors that dominate the performance of the codes.

Mixing LBP and ANS can solve trapping set errors without incurring the previously mentioned ANS greedy errors. We show experimentally that these strategies perform very well for 5 different short-blocklength codes. Furthermore, mixed strategies have a lower complexity than ANS since LBP is simpler than ANS. Also, we propose LC-ANS as another IDS strategy that performs as well as ANS while having a lower complexity.

Finally, a parallel implementation of ANS (P-ANS) was shown to perform nearly as well as ANS, making this IDS attractive for practical implementations, but it is not expected that IDS will be lower in implementation complexity than LBP. The main benefit of IDS ultimately is that it provides better ultimate BER and FER performance.

The improvement in performance of IDS strategies was shown for a variety of codes with different blocklengths and rates. However, these improvements come at the cost of a significant increase in complexity due to the computations needed to select the messages for propagation. The extent of the complexity cost requires further investigation in the context of specific implementations.

The ideas presented in this work may be extended to other communication solutions that use loopy BP, such as turbo codes, turbo-equalization, iterative demodulation and decoding of high-order constellations, among others. The extensions of the IDS strategies may also prove beneficial for loopy-BP solutions to problems outside the communications field.

References

[1] R.J. McEliece, D.J.C. MacKay, and Jung-Fu Cheng. "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE J. Sel. Areas Commun.*, vol. 16, pp. 140-152, Feb. 1998.

[2] F. Kschischang, B. J. R. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498-519, Mar. 2001.

[3] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam. "High Throughput Low-Density Parity-Check Decoder Architectures," in *Proc. IEEE GLOBECOM*, pp. 3019-3024, San Antonio, TX, Nov. 2001.

[4] M.M. Mansour and N.R. Shanbhag. "High-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 11, pp. 976-996, Dec. 2003.

[5] J. Zhang and M. Fossorier. "Shuffled belief propagation decoding," in *Proc. 36th Annual Asilomar Conf. Signals, Syst. Comput.*, pp. 8-15, Nov. 2002.

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

[7] D. Hocevar. "A reduced complexity decoder architechture via layered decoding of LDPC codes," in *Proc. Signal Process. Syst. SIPS 2004*, pp. 107-112, Oct. 2004.

[8] E. Sharon, S. Litsyn, and J. Goldberger. "An efficient message-passing schedule for LDPC decoding," in *Proc. 23rd IEEE Convention Electrical Electronics Engineers Israel*, pp. 223-226, Sept. 2004.

[9] E. Sharon, S. Litsyn, and J. Goldberger. "Efficient message-passing schedule for LDPC decoding," *IEEE Trans. Inf. Theory*, vol. 53, pp. 4076-4091, Nov. 2007.

[10] P. Radosavljevic, A. de Baynast, and J.R. Cavallaro. "Optimized message passing schedules for LDPC decoding," in *Proc. 39th Asilomar Conf. Signals, Syst. Comput.*, pp. 591-595, 2005.
[11] F. Guilloud, E. Boutillon, J. Tousch, and J.L. Danger. "Generic description and synthesis of LDPC decoder," *Accepted for publication, IEEE Trans. Commun.*.
[12] A. I. Vila Casado, M. Griot, and R. Wesel. "Overcoming LDPC trapping sets with informed scheduling," in *Inf. Theory Applications Workshop*, UCSD, San Diego, CA, Jan. 2007.
[13] A. I. Vila Casado, M. Griot, and R. Wesel. "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proc. IEEE ICC 2007*, Glasgow, Scotland, June 2007.
[14] Valentin Savin. "Iterative ldpc decoding using neighborhood reliabilities," Jan. 15th 2007.
[15] G. Elidan, I. McGraw, and D. Koller. "Residual belief propagation: informed scheduling for asynchronous message passing," in *Proc.* 22^{nd} *Conf. Uncertainty Artificial Intelligence*, MIT, Cambridge, MA, July 2006.
[16] N. Wiberg. "Codes and decoding on general graphs," Ph.D. Dissertation, Department of Electrical Engineering, Linkoping University, Linkoping, Sweden. 1996.
[17] M. Fossorier, M. Mihaljevic, and H. Imai. "Reduced complexity iterative decoding of low density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673-680, May 1999.
[18] *IEEE P802.11n/D1.05 October 2006, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - Enhancements for Higher Throughput (Draft).*
[19] D. MacKay and M. Postol. "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," *Electron. Notes Theoretical Computer Science*, vol. 74, 2003.
[20] T. Richardson. "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. Commun.*, Monticello, IL, 2003.
[21] G. A. Margulis. "Explicit constructions of graphs without short cycles and low-density codes," *Combinatorica 2*, vol. 1, pp. 71-78, 1982.
[22] T. Tian, C. Jones, J. Villasenor, and R. Wesel. "Avoidance of cycles in irregular LDPCC construction," in *IEEE Trans. Commun.*, Aug. 2004.
[23] A. Ramamoorthy and R. D. Wesel. "Construction of short block length irregular LDPCCs," in *Proc. IEEE ICC 2004*, Paris, France, June 2004.
[24] Xiao Yu Hu, Evangelos Eleftherioua, and Dieter Michael Arnold. "Progressive edge-growth tanner graphs," in *Proc. IEEE GLOBECOM*, pp. 995-1001, San Antonio, TX, Nov. 2001.
[25] M. Rovini, F. Rossi, P. Ciao, N. L'Insalata, and L. Fanucci. "Layered decoding of non-layered LDPC codes," in *Proc. 9th EUROMICRO Conf. Digital Syst. Design*, pp. 537-544, Aug. 2006.

Andres I. Vila Casado received his B. S. in electrical engineering from the Politecnico di Torino, Turin, Italy in 2002. He received his M. S. and Ph. D. degrees in electrical engineering from the University of California, Los Angeles in 2004 and 2007 respectively. At UCLA and at Politecnico di Torino he conducted research on communication theory with a focus on channel coding and information theory. He is currently a Research Scientist at Mojix, Inc. where he conducts research on physical layer communications and Bayesian estimation for RFID applications.

Miguel Griot received his B.S. degree from the Universidad de la Republica, Uruguay, in 2003, his of California at Los Angeles, in 2004 and 2008, respectively, all in electrical engineering. He is currently with Qualcomm Inc., San Diego, CA. His research interests include wireless communications, channel coding, information theory, multiple access channels, and broadcast channels.

Richard D. Wesel is a Professor with the UCLA Electrical Engineering Department and is the Associate Dean for Academic and Student Affairs for the UCLA Henry Samueli School of Engineering and Applied Science. He joined UCLA in 1996 after receiving his Ph.D. in electrical engineering from Stanford. His B.S. and M.S. degrees in electrical engineering are from MIT. His research is in the area of communication theory with particular interest in channel coding. He has received the National Science Foundation (NSF) CAREER Award, an Okawa Foundation award for research in information and telecommunications, and the Excellence in Teaching Award from the Henry Samueli School of Engineering and Applied Science. He has authored or co-authored over a hundred conference and journal publications.