Asynchronous Decoding of LDPC Codes over BEC

Saeid Haghighatshoar*, Amin Karbasi†, Amir Hesam Salavati‡
*Department of Telecommunication Systems, Technische Universität Berlin, Germany, †School of Engineering and Applied Sciences, Yale, USA, ‡School of Computer and Communication Sciences, EPFL, Switzerland
Emails: saeid.haghighatshoar@tu-berlin.de, amin.karbasi@yale.edu, hesam.salavati@epfl.ch

***Abstract*—LDPC codes are typically decoded by running a *synchronous* message passing algorithm over the corresponding bipartite factor graph (made of variable and check nodes). More specifically, each synchronous round consists of 1) updating all variable nodes based on the information received from the check nodes in the previous round, and then 2) updating all the check nodes based on the information sent from variable nodes in the current round. However, in many applications, ranging from message passing in neural networks to hardware implementation of LDPC codes, assuming that all messages are sent and received at the same time is far from realistic.**

In this paper, we investigate the effect of asynchronous message passing on the decoding of LDPC codes over a Binary Erasure Channel (BEC). We effectively assume that there is a random delay assigned to each edge of the factor graph that models the random propagation delay of a message along the edge. As a result, the output messages of a check/variable node are also asynchronously updated upon arrival of a new message in its input. We show, for the first time for BEC, that the asymptotic performance of the asynchronous message passing is fully characterized by a fixed point integral equation that takes into account both the *temporal* and the *spatial* features of the factor graph. Our theoretical result is reminiscent of the fixed point equation in traditional BP decoding. Also, our simulation results show that asynchronous scheduling reduces decoding time compared to the traditional BP in certain cases in the finite block-length regime.

I. Introduction

LDPC codes are a class of codes with a sparse parity check matrix and a sparse bipartite graph representation [1]. They can achieve a significant fraction of the channel capacity under a relatively low-complexity iterative belief propagation (BP) algorithm [2], [3]. This algorithm relies on a graph-based representation of the code, and it can be seen as message passing between variable and check nodes in the underlying bipartite graph. The order in which the messages are passed between these nodes is referred to as an updating rule or a *schedule*. The simplest message-passing schedule is a version of the so-called flooding schedule [4], in which in each iteration all the variable nodes, and subsequently all the check nodes, pass new messages to their neighbors. It has been empirically observed that the performance of the message passing algorithm might highly depend on the schedule [5]. Consequently, there has been several attempts to find a schedule that guarantees a better performance, in terms of finite-length bit error rate, convergence speed, etc., than the flooding schedule.

In particular, Mao and Banihashemi [6] used a probabilistic variant of the flooding schedule by taking into account the loop structure of the code's graph. The proposed schedule was shown to reduce the error floor and the number of undetected errors. Later, Zhang and Fossorier [7] as well as Kfir and Kanter [8] proposed a serial decoding schedule based on updating the variable node messages in a serial manner (it can be seen as a shuffling of the flooding schedule). Empirical simulations shows that a serial decoding schedule has a faster convergence than the flooding schedule. Similarly, in [9] a new dual schedule was proposed based on the serial update of check node messages which is more efficient than the one proposed in [7], [8].

Going beyond the realm of iterative coding/decoding, message passing algorithms, due to their low computational complexity, have been extensively used as an effective strategy for inference over graphical models. These methods, which originated with the simple loopy belief propagation (BP) algorithm of Pearl [10], have been the focus of much research; many extensions have been proposed, and much analysis has been done on their convergence [11], and in particular on its dependence on the message scheduling. The closest work, in spirit, to our efforts, is *residual belief propagation* (RBP) algorithm [12] where messages are updated sequentially (one message per iteration) with an additional possibility of random selection of messages in each iteration. Since sequential scheduling explores a random local neighborhood of every node in the graph (rather than a fixed-depth neighborhood as it is done in synchronized scheduling), it tends to converge faster to a fixed point than a synchronized schedule.

Scheduling aside, asynchronous message algorithms also naturally arise when running iterative decoding algorithms in VLSI circuit, also known as the *analog decoding* problem [13]. It has been shown that a relaxed version of BP for analog LDPC codes actually outperforms the digital synchronous version in BSC and AWGN channels [14].

In this paper, we mainly focus on the coding scenario with a given ensemble of LDPC codes over a Binary Erasure Channel (BEC) of a given erasure parameter ϵ. We modify the underlying graph ensemble by adding a *temporal feature*. More precisely, we add independent propagation delays to the edges of the graph with each delay being sampled from an arbitrary but fixed probability distribution p_d over $\mathbb{R}_+$. Such delays can be introduced artificially or they may simply be part of the underlying application (e.g., neural networks, FPGA or VLSI implementation of LDPC decoders). Based on these delay parameters, we propose a

simple asynchronous scheduling for message passing. The proposed scheduling is very similar to flooding scheduling with the difference that messages travel along the edges of the graph with a propagation delay that is assigned to each edge. Once a message arrives at a variable/check node all the outgoing messages are updated and start propagating to the neighboring nodes. There are some similarities between our method and sequential update scheduling proposed in [7], [8]. At each time instance, the outgoing message of a variable/check node depends on the information in a random local neighborhood rather than a fixed-depth one resulted from the flooding scheduling. Also note that by changing the assigned delay parameters, one can completely change the explored random neighborhood. We analyze the asymptotic performance of our asynchronous message passing algorithm in terms of the delay distribution p_d and the graph structure. Moreover, through simulations, we asses numerically the performance of our asynchronous scheduling in terms of the convergence speed and correcting errors.

II. Notation and Problem Statement

In this section, we first define the ensemble of asynchronous LDPC bipartite graphs. For this ensemble, we then introduce the asynchronous message passing algorithm that aims to infer the information bits of the transmitted code over a $\mathsf{BEC}(\epsilon)$.

A. Ensemble of Asynchronous LDPC Bipartite Graphs

We follow the standard notation used in [15] to introduce the ensemble of LDPC bipartite graphs. An element from this ensemble is generated by constructing a Tanner graph on n variable nodes and m check nodes as follows. Let the number of variable nodes of degree i be Λ_i, thus, $\sum_i \Lambda_i = n$. Similarly, let P_j denote the number of check nodes of degree j, where $\sum_j P_j = m$. Then, each node is associated with a number of sockets equal to its degree. An element of the LDPC ensemble, denoted by $\text{LDPC}(\Lambda, P)$, is generated by matching the check and variable sockets according to a uniformly random permutation. Note that the elements of $\text{LDPC}(\Lambda, P)$ are nothing but bipartite graphs. Based on the ensemble of $\text{LDPC}(\Lambda, P)$ graphs, we can build the ensemble of asynchronous $\text{LDPC}(\Lambda, P)$ graphs by assigning temporal delay parameters to the edges as follows.

Definition 1. Let p_d be a probability distribution over $\mathbb{R}_+$. Elements of the asynchronous ensemble $\text{ALDPC}(\Lambda, P, p_d)$ are generated as follows: we pick an element from $\text{LDPC}(\Lambda, P)$ and assign i.i.d. delay parameters to the edges of the graph according to p_d .

We call the variable-check adjacency of a graph generated from the ensemble $\text{ALDPC}(\Lambda, P, p_d)$ the *spatial feature* of the ensemble. The delay parameters assigned to the edges of the graph constitute the *temporal feature* of the ensemble. Similar to the $\text{LDPC}(\Lambda, P)$, we can construct a code from a graph in the ensemble $\text{ALDPC}(\Lambda, P, p_d)$ by simply dropping the delay parameters of the edges. Recall that to assign a code to a specific instance of the ensemble $\text{LDPC}(\Lambda, P)$, we form an m-by-n parity check matrix H with $\{0, 1\}$ components, where $H_{i,j} = 1$ if and only if the j-th variable node is connected to the i-th check node. Hence, the code structure only depends on the spatial feature of the underlying graph. As we will explain later, the temporal feature plays a prominent role when we consider inference via an iterative message passing algorithm.

Let us also define the variable and check degree distributions for a graph from $\text{ALDPC}(\Lambda, P, p_d)$ by

$$\Lambda(x) = \sum_{i=1}^{\mathsf{l}_{\max}} \Lambda_i x^i, P(x) = \sum_{j=1}^{\mathsf{r}_{\max}} P_j x^j, \tag{1}$$

and their normalized versions by

$$L(x) = \frac{\Lambda(x)}{\Lambda(1)}, \; R(x) = \frac{P(x)}{P(1)}. \tag{2}$$

where $\mathsf{l}_{\max}$ and $\mathsf{r}_{\max}$ denote the maximum variable and check degrees. For the asymptotic analysis carried out in this paper, it is more convenient to take an edge perspective defined as follows:

$$\lambda(x) = \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{L'(x)}{L'(1)}, \; \rho(x) = \frac{P'(x)}{P'(1)} = \frac{R'(x)}{R'(1)}. \tag{3}$$

B. Inference via Asynchronous Message Passing

Recall that to decode the information bits of a graph sampled from $\text{LDPC}(\Lambda, P)$ whose variables are observed via a BEC, one typically uses Belief Propagation (BP) by passing messages/beliefs between variable and check nodes according to an arbitrary schedule. When the underlying bipartite graph is a tree, and as long as the messages get updated often enough, they will all converge to the right marginal distribution in a finite number of iterations. In contrast, when there are cycles in the graph, there is generally no guaranty that BP approaches to the true marginal distribution. However, if the graph is sparse, meaning that it has a locally tree-like neighborhood around every check and variable node, then BP decoding provides a very good approximation. It is observed that the performance (rate of convergence, etc.) highly depends on the message scheduling. In its simplest form, namely, synchronous BP, messages are sent in synchronous rounds. In this case, the asymptotic decoding performance (as n gets large) of the BP algorithm is fully characterized by a density evolution equation (DE). More precisely, let us define $\mathsf{de}(x) \triangleq \epsilon\lambda(1 - \rho(1 - x))$, where ϵ is the erasure probability of the channel. Let x_ℓ be the fraction of undecoded code-bits after ℓ round of message passing between the variable and check nodes, then it was shown that x_ℓ satisfies the density evolution iteration $x_\ell = \mathsf{de}(x_{\ell-1})$, for $\ell \geq 1$ with the initialization $x_0 = 1$ [2].

To introduce our asynchronous message passing algorithm, consider a graph sampled from $\text{ALDPC}(\Lambda, P, p_d)$. We first start from check nodes and assume that all their messages are initialized to 1. This is intuitively the belief of a check node about its neighbors (i.e., variable nodes) being an erasure. These messages start propagating along the edges

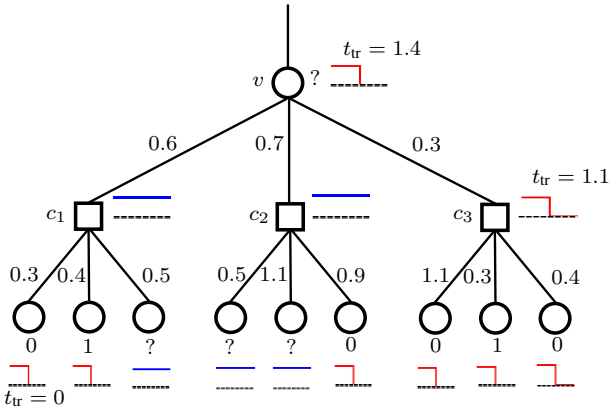


Fig. 1: Illustration of the Asynchronous Message Passing for a tree of depth 2. The numbers on the edges show the associated propagation delay of the edge.

of the graph. The time it takes for a message to reach the variable node is given by the assigned propagation delay of the edge. On the variable side, every variable node upon receiving a message updates all of its outgoing messages along all its edges (except the edge from which it receives the message). Note that messages no longer reach variable and check nodes at the same time. Also, nodes (being check or variable) do not wait for all the incoming messages in order to send out the outgoing messages; they act as soon as they receive an incoming message.

Figure 1 illustrates a simple example of the asynchronous message passing algorithm for a random graph from the ensemble ALDPC(Λ, P, p_d) for a local neighborhood of a variable node v of depth 2. Before the start of the algorithm, all the messages propagating along the edges are equal to 1 (erasure belief about the neighboring nodes). At time $t = 0$, the channel observations are revealed. The output message for those variables that are not erased by the channel transitions immediately from 1 to 0 and propagates along the outgoing edges to the neighboring check nodes. Looking at check nodes, it can be seen that c_3 is the only check node among c_1, c_2, c_3 whose outgoing message can make a transition and this happens once all the 1-to-0 erasure transition messages arrive from the neighboring variable nodes, which happens at $t_{\text{tr}} = 1.1$. Variable node v at the top of the tree has an erasure message from the channel, thus, its outgoing message stays 1 at $t = 0$. Since c_1 and c_2 always send erasure message 1, v can only be decoded if it receives a 1-to-0 transition message from c_3, which happens at $t = 1.4$ as seen from the figure.

Asynchronous message passing is more suitable for many real-world applications. For instance, when the graph is very large such that nodes reside on different machines, communicating between devices takes random delays. Similarly, for hardware implementation of LDPC code (on VLSI or FPGA) there is always a random delay (dictated by the physical wiring) between sending a message from one node and receiving it by another. The same is true for neural networks. Even though many models of neural networks assume synchronous communication among neurons, such models are far from reality. We will show that there is no need to assume synchrony to obtain analytical guarantees.

III. Asymptotic Performance Analysis

In this section, we analyze the performance of the asynchronous message passing over BEC(ϵ). The analysis is reminiscent of the traditional BP algorithm, which in turn leads to a density evolution integral equation. Let us recall the computation graph ensemble $\vec{C}_\ell(n, \lambda, \rho)$ of height ℓ from an edge perspective. An element from $\vec{C}_\ell(n, \lambda, \rho)$ is drawn by (i) choosing a random element G from ALDPC(Λ, P, p_d), (ii) choosing a random edge e in G (note that e is incident to a variable and a check node), and (iii) letting T be the subgraph of G induced by depth-$(2\ell + 1)$ directed neighborhood of e from the variable node side. It is known that for any fixed $\ell > 0$, the probability that the computation graph is a tree up to height ℓ approaches 1 as n tends to infinity [2]. Hence, this result implies that for a fixed ℓ and as n grows large, the error probability is equal to that observed on a tree. Let $\vec{\mathcal{T}}_\ell(\lambda, \rho)$ denote the ensemble of computation trees with height-ℓ from the edge perspective. A sample T is generated from this ensemble by (i) choosing a variable node of degree d according to $\lambda(x)$, (ii) attaching $d-1$ check nodes whose degrees are drawn i.i.d. according to $\rho(x)$, and (iii) attaching to each open check edge a random element of $\vec{\mathcal{T}}_{\ell-1}(\lambda, \rho)$. To start the recursive definition, $\vec{\mathcal{T}}_0(\lambda, \rho)$ is drawn from $\lambda(x)$. We also assume that i.i.d. propagation delays with distribution p_d are assigned to all the edges of the resulting tree.

Definition 2. For a $\mathsf{T} \in \vec{\mathcal{T}}_\ell(\lambda, \rho)$, we define $\mathsf{E}(\mathsf{T}, \epsilon, t)$ as the output erasure indicator function of the root variable node of T at time t of the asynchronous decoding when all the variable nodes of T are observed through a BEC(ϵ).

A sample realization of E is shown in Figure 1 for a simple tree of height $\ell = 1$. Note that $\mathsf{E}(\mathsf{T}, \epsilon, t)$ is a $\{0, 1\}$-valued step function with 1 showing that the variable node is not still decoded, i.e., it is in the erasure state. As we start the algorithm at $t = 0$, as an initialization, we assume that $\mathsf{E}(\mathsf{T}, \epsilon, t) = 1$ for $t < 0$. Once a variable node is decoded, it remains decoded, thus, E has at most one 1-to-0 transition for $t \geq 0$ whose time depends on the erasure pattern of the variables nodes and the propagation delay of the edges of T. As a very simple case, if the variable node is not erased by the channel, the jump location is $t = 0$, i.e., $\mathsf{E}(\mathsf{T}, \epsilon, t) = 0$ for $t \geq 0$, implying that it is immediately decoded at $t = 0$. We also define the average erasure function as follows.

$$f_\ell(\epsilon, t) \triangleq \sum_{\mathsf{T}} \mathbb{P}(\vec{\mathcal{T}}_\ell(\lambda, \rho) = \mathsf{T}) \mathsf{E}(\mathsf{T}, \epsilon, t), \tag{4}$$

where $f_\ell(\epsilon, t)$ is the erasure probability of a typical variable node at time t based on all the information received from a height-ℓ tree neighborhood, and where the average is taken over the tree structure (spatial feature) as well as the random propagation delays (temporal feature) of T.

For a real-valued function $g(t)$, we define its average w.r.t the delay as $\overline{g(t)} = \mathbb{E}_{p_d}\{g(t-d)\}$. Throughout this paper, we assume that p_d is supported on positive reals. In particular, $p_d(\{0\}) = 0$. This avoids an avalanche of decoding events over the graph during a finite time interval.

Theorem 1 (Finite Depth DE Equation)**.** *The asynchronous decoding of an LDPC ensemble, with degree distributions (λ, ρ), satisfies the following density evolution:*

$$f_\ell(\epsilon, t) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{f_{\ell-1}(\epsilon, t)})}), t \geq 0, \quad (5)$$

$$f_\ell(\epsilon, t) = 1, t < 0,\ \ell \geq 0 \quad (6)$$

$$f_0(\epsilon, t) = \epsilon, t \geq 0, \quad (7)$$

where ℓ denote the message passing iteration between the variable and check nodes.

Theorem 1 characterizes the behavior of the average erasure probability for a fixed ℓ. In the asymptotic regime, what we are really interested is $f_\infty(\epsilon, t) \triangleq \lim_{\ell\to\infty} f_\ell(\epsilon, t)$, which takes into account all the asynchronous messages received from any arbitrary depths in the graph. The following theorem fully characterizes $f_\infty(\epsilon, t)$ as we run the asynchronous message passing.

Theorem 2 (Infinite Depth DE Equation)**.** $f_\infty(\epsilon, t)$ *satisfies the following fixed point integral equation:*

$$f_\infty(\epsilon, t) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{f_\infty(\epsilon, t)})}) \quad (8)$$

with the initialization $f_\infty(\epsilon, t) = 1, t < 0$.

To better understand the implications of Theorem 2, let us consider a simple example. The synchronous message passing algorithm is the one with $p_d = \mathbf{1}_{\{d=\frac{1}{2}\}}$, which corresponds to half a unit of delay per edge. In this case, the fixed point integral equation can be simply written as

$$f_\infty(\epsilon, t) = \epsilon\lambda(1 - \rho(1 - f_\infty(\epsilon, t-1))). \quad (9)$$

It is not difficult to see that the resulting solution is a right-continuous stair-case function with

$$f_\infty(\epsilon, t) = f_\infty(\epsilon, k) \quad t \in [k, k+1),\ k \in \mathbb{Z}_+.$$

From Eq. (9), for $k \in \mathbb{Z}_+$, one obtains

$$f_\infty(\epsilon, k) = \epsilon\lambda(1-\rho(1-f_\infty(\epsilon, k-1))) = \mathsf{de}(f_\infty(\epsilon, k-1)),$$

with the initialization $f_\infty(\epsilon, -1) = 1$. The above equation is nothing but the traditional density evolution, where each round of the message passing takes 1 unit of time.

The function $f_\infty(\epsilon, t)$ characterizes the performance of the asynchronous message passing decoding in the asymptotic regime as a function of time. The decoding is successful if $f_\infty(\epsilon) \triangleq \lim_{t\to\infty} f_\infty(\epsilon, t) = 0$, where the decoding time depends on how fast the function approaches 0.

Theorem 3. *Let ALDPC(Λ, P, p_d) be the code ensemble with a degree distribution (λ, ρ). For a given channel erasure parameter ϵ, let $\mathsf{sync}(\epsilon)$ be the largest fixed point of the synchronized DE equation $\mathsf{de}(x) = \epsilon\lambda(1 - \rho(1 - x))$ in $[0, 1]$. Then $f_\infty(\epsilon) = \mathsf{sync}(\epsilon)$.*

Theorem 3 essentially implies that for large block-lengths the asynchronous message passing has a behavior similar to the traditional synchronous one. In the next section, we use numerical simulations to show that the same phenomenon also occurs in the finite block-length regime.

IV. Simulation Results

In this section, we first numerically validate our theoretical analysis for the asymptotic regime and then show similar trends also hold for the performance of the asynchronous decoding in the finite length regime, with improvements in decoding time.

A. Asymptotic Regime

Solution of the fixed point equation: We numerically solve the fixed point integral equation for an exponential delay with mean $\frac{1}{2}$, i.e., $p_d(x) = 2e^{-2x}\mathbf{1}_{\{x\geq 0\}}$. This corresponds to an average of 0.5 time unit propagation delay along the edges of the graph. We use the $(4, 8)$-regular ensemble with an approximate BP threshold of $\epsilon_{\text{BP}} = 0.38$. The resulting asymptotic curve is shown in Figure 2a for two different values of ϵ, one above and one below ϵ_{BP}.

Comparison with synchronous BP: In Figures 2b and 2c, we compare the performance of the synchronous and asynchronous message passing algorithms in terms of the decoding time for two different values of ϵ: $\epsilon = 0.3 \ll \epsilon_{\text{BP}}$ and $\epsilon = 0.37 \simeq \epsilon_{\text{BP}}$. As we see, asynchronous message passing converges to the right codeword much faster.

B. Finite-Length Regime

In the finite-length regime, we again consider the $(4, 8)$-regular ensemble where we fix the number of variable nodes to $n = 128$ and the number of check nodes to $m = 64$[1]. We also consider 3 message passing algorithms:

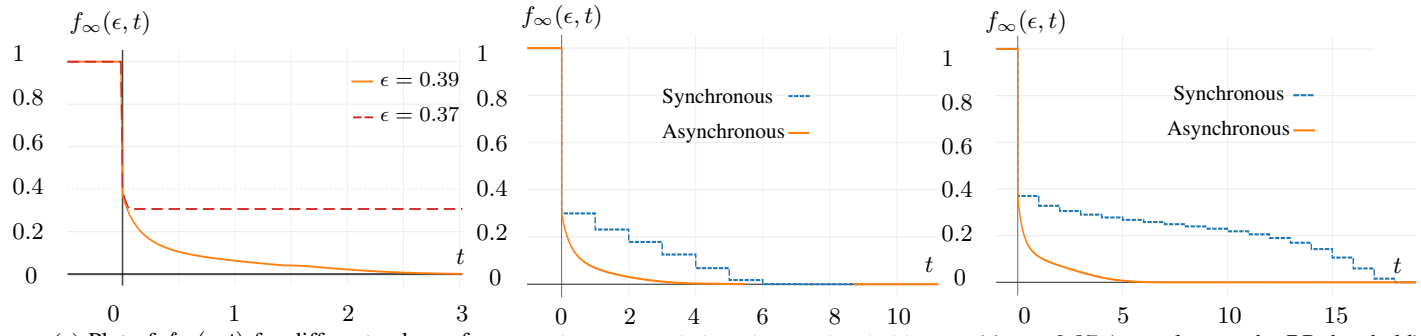
Standard BP: Each link is assigned a fixed delay of 0.5 unit time. Hence, for every message, it takes exactly 0.5 unit time to reach a node. We also call this algorithm synchronous decoder.

Asynchronous Decoding: To send a message along an edge, an i.i.d random delay from an exponential distribution with mean 0.5 is sampled and assigned to the edge. Hence, messages reach nodes in different times.

Jittered Asynchronous Decoding: Before decoding, the delay of each link is sampled i.i.d. from an exponential distribution with mean 0.5. During the decoding, the delay of an edge slightly varies around its initial sampled value.

In all cases, the average delay across all links is 0.5 unit time. To model the delay variations of link in jittered asynchronous BP, we sample the variations from a uniform distribution with mean 0 and maximum value of 0.05 (in time unit). Figure 3 illustrates the Bit Error Rate (BER) for the three algorithms. The figure clearly shows that even in the finite-length regime, the performance of the asynchronous decoder closely follows the synchronous one.

[1]The simulation code is available at rr.epfl.ch/paper/HKS2015.



(a) Plot of $f_\infty(\epsilon, t)$ for different values of ϵ (b) $\epsilon = 0.3$ (below the BP threshold) (c) $\epsilon = 0.37$ (very close to the BP threshold)

Fig. 2: Comparison of the performance of the synchronous (traditional) and asynchronous (exponential delay distribution) message passing for different values of ϵ and for a $(4, 8)$-regular ensemble with an exponential delay distribution with average $\frac{1}{2}$ (one round of message passing takes on average 1 unit of time.)

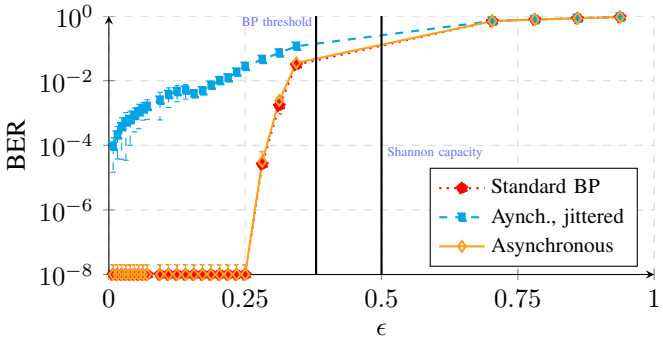


Fig. 3: BER for three different decoders over an erasure channel. Shannon's limit and the standard BP decoding threshold for a (4,8)-regular ensemble are marked as well.

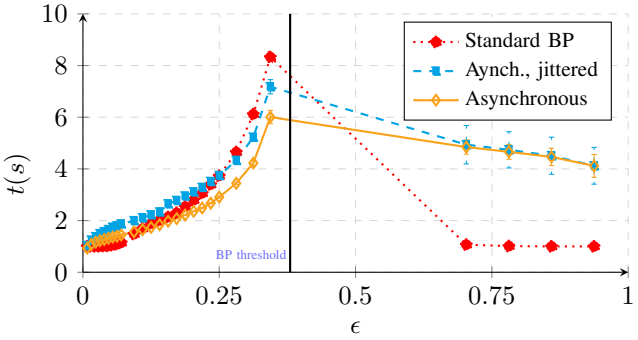


Fig. 4: The (virtual) *time* of the last decoding event processed by the three different decoding methods.

Next, we investigate the effect of decoding methods (i.e., synchronous vs. asynchronous) on the number of iterations performed by the decoder. The complete set of results are given in the extended version of this paper [16]. Note that since the synchronous decoder rapidly identifies whether or not it can eliminate erasures, it usually spends fewer iterations (with more failed status) than the asynchronous decoder. In contrast, the decoding *time*[2] seems to be generally lower for the asynchronous decoder in the operational regime (below the BP threshold), as shown in Fig. 4.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Tran. Inf. Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Tran. Inf. Theory*, vol. 47, no. 2, pp. 599–618, 2001.

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

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

[5] G. D. Forney Jr, "On iterative decoding and the two-way algorithm," in *Proc. Int. Symp. Turbo Codes and Related Topics*, 1997, pp. 12–15.

[2]Note that this is the time of the last event processed by the decoder (taking into account the link delays) and differs from the *CPU time* when running simulations. Hence, decoding time is more relevant to the cases where message passing is performed in a distributed manner, i.e., separately located nodes collectively trying to minimize the error probability.

[6] Y. Mao and A. H. Banihashemi, "Decoding low-density parity-check codes with probabilistic schedule," in *Proc. IEEE PACRIM*, vol. 1. IEEE, 2001, pp. 119–123.

[7] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Asilomar Conference on Signals, Systems and Computers*, vol. 1. IEEE, 2002, pp. 8–15.

[8] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A: Statistical Mechanics and its Applications*, vol. 330, no. 1, pp. 259–270, 2003.

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

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[11] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Tran. Inf. Theory*, vol. 49, no. 5, pp. 1120–1146, 2003.

[12] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," *arXiv preprint arXiv:1206.6837*, 2012.

[13] H.-A. Loeliger, F. Tarkoy, F. Lustenberger, and M. Helfenstein, "Decoding in analog vlsi," *IEEE Communications Magazine*, vol. 37, no. 4, pp. 99–101, 1999.

[14] S. Hemati and A. H. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (ldpc) codes," *IEEE Tran. Comm.*, vol. 54, no. 1, pp. 61–70, 2006.

[15] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[16] S. Haghighatshoar, A. Karbasi, and A. H. Salavati, "Asynchronous Scheduling for BP decoding of LDPC codes over BEC," 2015. [Online]. Available: http://rr.epfl.ch/paper/HKS2015

APPENDIX

A. Proof of Theorem 1

First note that the algorithm starts at $t = 0$, thus, before the start of the algorithm no message is received (even the erasure message from the channel). Hence, for every variable node and its tree neighborhood T of depth ℓ, we have $\mathsf{E}(\mathsf{T}, t) = 1$ for $t < 0$. Thus, its average is also 1, which implies that $f_\ell(\epsilon, t) = 1$ for every $\ell \geq 0$ and $t < 0$. Moreover, for $\ell = 0$, we have a neighborhood of height 0, which contains only the erasure message received from the channel, which we assume to happen immediately at $t = 0$. This implies that for a variable node v, the corresponding erasure function $\mathsf{E}(\mathsf{T}, \epsilon, t) = \mathbf{1}_{\{\text{erasure in } v\}}$ for $t \geq 0$. Hence, taking the expectation value, we obtain $f_0(\epsilon, t) = \epsilon$ for $t \geq 0$. This completes the initialization step. Also notice that, as we assume that $p_d(\{0\}) = 0$, at $t = 0$ no message is delivered except the erasure message from the channel and by a similar argument, one can show that $f_\ell(\epsilon, 0) = \epsilon$ for $\ell \geq 1$. This can also be obtained from the density evolution iteration.

For the rest of the proof, let us call the top variable node v and let $c_1, c_2, \ldots, c_K$ be the check nodes connected to v, where K is a random number with $\mathbb{P}(K = k) = \lambda_{k+1}$. Let $A_1(t), A_2(t), \ldots, A_K(t)$ be the erasure messages at the output of these check nodes at time t of the message passing. Recall that $A_i(t)$ are stochastic step functions with $A_i(t) = 1$ for $t < 0$, which have at most one 1-to-0 transition. Also notice that $A_i(t)$ are independent from each other. We also define $a_i(t) = \mathbb{E}\{A_i(t)\}$, where the average is taken over all the randomness of the underlying subtrees. From the symmetry, it results that $a_i(t) = a_{i'}(t) = a(t)$ for all $i \neq i'$. The erasure function for the node v, can be written as

$$B(t) = \mathbf{1}_{\{\text{channel erasure observation for } v\}} \prod_{i=1}^{K} A_i(t - d_i), \quad (10)$$

where d_i is the propagation delay on the edge between v and c_i. One can check that d_i and $A_i(t)$ for $i = 1, 2, \ldots, K$, the channel erasure and the number of neighboring checks K are totally independent of one another. Hence, taking the average, we obtain the average erasure output of the node v for a height-ℓ tree neighborhood $f_\ell(\epsilon, t) = \mathbb{E}\{B(t)\}$ as follows:

$$f_\ell(\epsilon, t) = \epsilon \sum_{k=0} \lambda_{k+1} \mathbb{E}\{\prod_{i=1}^{k} A_i(t - d_i)\} \quad (11)$$

$$= \epsilon \sum_{k=0} \lambda_{k+1} \prod_{i=1}^{k} \mathbb{E}_{p_d}\{a_i(t - d_i)\} \quad (12)$$

$$= \epsilon \sum_{k=0} \lambda_{k+1} \overline{a(t)}^k = \epsilon \lambda\Big(\overline{a(t)}\Big), \quad (13)$$

where $\overline{a(t)} = \mathbb{E}_{p_d}\{a(t - d)\}$ denotes the delay-averaged version of $a(t)$. To compute $a(t)$, let us consider the check node c_1. Apart from v, check node c_1 is connected to a random number of variable nodes $v_1, v_2, \ldots, v_M$, where $\mathbb{P}(M = m) = \rho_{m+1}$. Let $B_1(t), B_2(t), \ldots, B_m(t)$ be the erasure messages at the output of these variable nodes and let $b_j(t) = \mathbb{E}\{B_j(t)\}$, where the average is taken with respect to the corresponding subtrees of v_j and their corresponding delay parameters. From symmetry, it again results that for every $j \neq j'$,

$$b_j(t) = b_{j'}(t) = b(t) = f_{\ell-1}(\epsilon, t). \quad (14)$$

The erasure message at the output of c_1 can be written as $A_1(t) = 1 - \prod_{j=1}^{M}(1 - B_j(t - d'_j))$, where d'_j denotes the random propagation delay of the edge between c_1 and the variable node v_j. Note that $A_1(t)$ is also a $\{0, 1\}$-valued step function with $A_1(t) = 0$ whenever $B_j(t) = 1$ for all j, in which case all the variable nodes v_j send a successful decoding message to c_1, which implies that the variable node v can be successfully decoded. Check node c_1 forwards this message to the variable node v via the in-between edge.

One can check that M and d'_j, $B_j(t)$ for all $j = 1, 2, \ldots, M$ are completely independent from one another. Hence, we obtain

$$a(t) = a_1(t) = \mathbb{E}\{A_1(t)\} \quad (15)$$

$$= \sum_{m=0} \rho_{m+1} \mathbb{E}\{1 - \prod_{j=1}^{m}(1 - B_j(t - d'_j))\} \quad (16)$$

$$= \sum_{m=0} \rho_{m+1}(1 - \prod_{j=1}^{m}(1 - \mathbb{E}_{p_d}\{b_j(t - d'_j)\})\} \quad (17)$$

$$= 1 - \sum_{m=0} \rho_{m+1}(1 - \overline{b(t)})^m = 1 - \rho(1 - \overline{b(t)}). \quad (18)$$

This implies that $\overline{a(t)} = 1 - \overline{\rho(1 - \overline{b(t)})}$. Thus, using (13) and (14), we obtain the desired result

$$f_\ell(\epsilon, t) = \epsilon \lambda(1 - \overline{\rho(1 - \overline{f_{\ell-1}(\epsilon, t)})}). \quad (19)$$

B. Proof of Theorem 2

We define the two-sides density evolution operator $\mathcal{O}$ as follows. This operator takes a function $g : \mathbb{R} \to [0, 1]$ and maps it onto $\tilde{g}(t) = \mathcal{O}(g) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{g(t)})})$. Hence, the density evolution in Theorem 1 can be written as

$$f_\ell(\epsilon, t) = \mathcal{O}(f_{\ell-1}(\epsilon, t)), t \geq 0, \quad (20)$$

$$f_\ell(\epsilon, t) = 1, t < 0. \quad (21)$$

We need the following propositions to prove Theorem 2.

Proposition 1 (Ordering Property)**.** *Let $I = [0, 1]$ and let $g_1, g_2 : \mathbb{R} \to I$ be functions such that $g_1(t) \leq g_2(t)$ for all $t \in \mathbb{R}$. Let $\tilde{g}_1 = \mathcal{O}(g_1)$ and $\tilde{g}_2 = \mathcal{O}(g_2)$ be the outputs of the two-sides density evolution operator. Then, $\tilde{g}_1(t) \leq \tilde{g}_2(t)$ for every $t \in \mathbb{R}$.*

Proof: First note that ρ and λ are increasing functions over I. One can also check that delay-averaging keeps the order, thus, $\overline{g_1(t)} \leq \overline{g_2(t)}$, for all $t \in \mathbb{R}$. Using the monotonicity of ρ, one has $0 \leq 1 - \rho(1 - \overline{g_1(t)}) \leq 1 - \rho(1 - \overline{g_2(t)}) \leq 1$. Taking the delay-average of both sides and using the monotonicity of λ, one obtains the result. ■

Proposition 2 (Delay Equivariance)**.** *Let $I = [0, 1]$ and let $g_1 : \mathbb{R} \to I$ be an arbitrary function and let $g_2(t) = g_1(t - \tau)$, where $\tau \in \mathbb{R}$ is an arbitrary number. Assume that $\tilde{g}_1 = \mathcal{O}(g_1)$ and $\tilde{g}_2 = \mathcal{O}(g_2)$. Then $\tilde{g}_1(t) = \tilde{g}_2(t - \tau)$.*

Proof: First note that

$$\overline{g_2(t)} = \mathbb{E}_{p_d}\{g_2(t - d)\} \tag{22}$$

$$= \mathbb{E}_{p_d}\{g_1(t - \tau - d)\} = \overline{g_1(t - \tau)}, \tag{23}$$

which is simply $\overline{g_1(t)}$ delayed by τ. This implies that $\rho(1 - \overline{g_2(t)})$ is a delayed copy of $\rho(1 - \overline{g_1(t)})$ by a delay of size τ. Applying the same argument as in Eq. (23), it results that $\rho(1 - \overline{g_2(t)})$ is a delayed version of $\rho(1 - \overline{g_2(t)})$ by a delay τ, and finally applying λ, we obtain the result. ∎

Proposition 3. *For every $t \in \mathbb{R}$ and for any $\ell \geq 1$, $f_\ell(\epsilon, t) \leq f_{\ell-1}(\epsilon, t)$.*

Proof: We prove using induction on ℓ. For $\ell = 0$, from the density evolution initialization, we have

$$f_0(\epsilon, t) = \begin{cases} 1 & t < 0, \\ \epsilon & t \geq 0. \end{cases} \tag{24}$$

As $f_0(\epsilon, t)$ takes values in $I = [0, 1]$, and $\rho, \lambda : I \to I$, are increasing functions, one can check from Eq. (5) that for $t \geq 0$,

$$f_1(\epsilon, t) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{f_0(\epsilon, t)})}) \leq \epsilon\lambda(1) = \epsilon, \tag{25}$$

which implies that $f_1(\epsilon, t) \leq f_0(\epsilon, t)$ for every $t \in \mathbb{R}$ (recall that $f_\ell(\epsilon, t) = 1$ for $t < 0$ and for every $\ell \geq 0$).

Now assume that for every $\ell \geq 1$, $f_\ell(\epsilon, t) \leq f_{\ell-1}(\epsilon, t)$. Using the monotonicity property of the two-sides density evolution operator $\mathcal{O}$ proved in Proposition 1, one immediately obtains that $f_{\ell+1}(\epsilon, t) \leq f_\ell(\epsilon, t)$ for $t \geq 0$. As $f_{\ell+1}(\epsilon, t) = f_\ell(\epsilon, t) = 1$ for $t < 0$, we obtain $f_{\ell+1}(\epsilon, t) \leq f_\ell(\epsilon, t)$ for all $t \in \mathbb{R}$. ∎

Proof of Theorem 2: Notice that $f_\ell(\epsilon, t) = 1$ for $t \leq 0$, thus $f_\infty(\epsilon, t) = 1, t \leq 0$, giving the appropriate initial condition. Also, from Proposition 3, for every $t \geq 0$, $f_\ell(\epsilon, t)$ is a decreasing function of ℓ, thus $f_\infty(\epsilon, t)$ is well defined for $t \geq 0$. From Theorem 1, for $t \geq 0$, we have

$$f_\ell(\epsilon, t) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{f_{\ell-1}(\epsilon, t)})}). \tag{26}$$

As $f_\ell(\epsilon, t)$ are bounded functions taking their values in $[0, 1]$, taking the limit as ℓ tends to infinity and applying the bounded convergence theorem, we obtain the desired result

$$f_\infty(\epsilon, t) = \epsilon\lambda(1 - \overline{\rho(1 - \overline{f_\infty(\epsilon, t)})}). \tag{27}$$

C. Proof of Theorem 3

We first need the following proposition.

Proposition 4. *For every $\ell \geq 0$, $f_\ell(\epsilon, t)$ is a decreasing function of t.*

Proof: We use induction on ℓ. For $\ell = 0$, from the density evolution initialization as in Eq. (24), one can check that as $\epsilon \in [0, 1]$, $f_0(\epsilon, t)$ is a decreasing function of t. Now assume that $f_\ell(\epsilon, t)$ is a decreasing function of t for some $\ell \geq 0$. This implies that for every $\tau > 0$ and $t \in \mathbb{R}$, $f_\ell(\epsilon, t) \leq f_\ell(\epsilon, t - \tau)$. Let $g(t) = \mathcal{O}(f_\ell(\epsilon, t))$. Using the monotonicity property form Proposition 1 and the delay-equivariance property from Proposition 2, one obtains that $g(t) \leq g(t - \tau)$, thus g is also a decreasing function of t. Now note that $f_{\ell+1}(\epsilon, t) = g(t)$ for $t \geq 0$, thus $f_{\ell+1}(\epsilon, t) \leq f_{\ell+1}(\epsilon, t - \tau)$ for $t \geq \tau$. Moreover, $f_{\ell+1}(\epsilon, t) \in [0, 1]$ and $f_{\ell+1}(\epsilon, t - \tau) = 1$ for $t < \tau$, which implies that $f_{\ell+1}(\epsilon, t) \leq f_{\ell+1}(\epsilon, t - \tau)$ for $t < \tau$. Hence, $f_{\ell+1}(\epsilon, t) \leq f_{\ell+1}(\epsilon, t - \tau)$ for all t and all $\tau > 0$, proving that $f_{\ell+1}(\epsilon, t)$ is also a decreasing function of t. ∎

Proof of Theorem 3: Note that from Proposition 4, $f_\infty(\epsilon, t) = \lim_{\ell\to\infty} f_\ell(\epsilon, t)$ is a limit of decreasing functions of t. Hence, it is a decreasing function of t and the limit $f_\infty(\epsilon) = \lim_{t\to\infty} f_\infty(\epsilon, t)$ is well defined. Now consider the fixed point integral equation in (8). Taking the limit as t tends to infinity and using the bounded convergence theorem (as in $\lim_{t\to\infty} \overline{f_\infty(\epsilon, t)} = f_\infty(\epsilon)$), we obtain the following equation:

$$f_\infty(\epsilon) = \epsilon\lambda(1 - \rho(1 - f_\infty(\epsilon))) = \mathsf{de}(f_\infty(\epsilon)), \tag{28}$$

which implies that $f_\infty(\epsilon)$ is also a fixed point of the synchronized density evolution function $\mathsf{de}(x)$.

Now, let $x_\ell = \mathsf{de}(x_{\ell-1})$, $\ell \geq 1$, be the DE iteration for the synchronized scheduling with the initialization $x_0 = 1$ converging to $\mathsf{sync}(\epsilon)$. Note that $x_\ell \in [0, 1]$. Moreover, as $\rho_0 = \rho(0) = 0$ (there is no fraction of edges connected to a degree 0 check node), $x_1 = \epsilon\lambda(1 - \rho(1 - x_0)) = \epsilon\lambda(1) = \epsilon$.

To prove the result, we first prove that $f_\ell(\epsilon, t) \geq x_{\ell+1}$, specially for $t \geq 0$. We use induction on ℓ. For $\ell = 0$, when $t < 0$, we have $f_0(\epsilon, t) = 1 \geq x_1$ and when $t \geq 0$, we have $f_0(\epsilon, t) = \epsilon \geq \epsilon = x_1$ and the result holds. Now assume that $f_\ell(\epsilon, t) \geq x_{\ell+1}$ and let us define $h(t) = x_{\ell+1}$ to be a constant function for all $t \in \mathbb{R}$. Thus, $f_{\ell+1}(\epsilon, t) \geq h(t)$ for all t. Using the monotonicity of the density evolution operator proved in Proposition 1, we obtain that for $t \geq 0$

$$f_{\ell+1}(\epsilon, t) = \mathcal{O}(f_\ell(\epsilon, t)) \geq \mathcal{O}(h(t))$$

$$= \epsilon\lambda(1 - \rho(1 - x_{\ell+1})) = \mathsf{de}(x_{\ell+1}) = x_{\ell+2},$$

which completes the induction. Taking the limit as ℓ tends to infinity, we obtain $f_\infty(\epsilon, t) \geq \mathsf{sync}(\epsilon)$. Finally, taking the limit as t tends to infinity, we obtain $f_\infty(\epsilon) \geq \mathsf{sync}(\epsilon)$. As $f_\infty(\epsilon)$ is itself a fixed point of $\mathsf{de}(x)$, and $\mathsf{sync}(\epsilon)$ is the largest fixed point of $\mathsf{de}(x)$, we obtain the desired result $f_\infty(\epsilon) = \mathsf{sync}(\epsilon)$.

D. Comparison of Decoding Time

The number of iterations performed by the three decoders is shown in Figure 5. Here, iteration is defined as the total number of messages exchanged over edges divided by the total number of edges in the parity check graph.

Figure 5 clearly shows that the number of iterations performed by asynchronous decoders are higher than that of

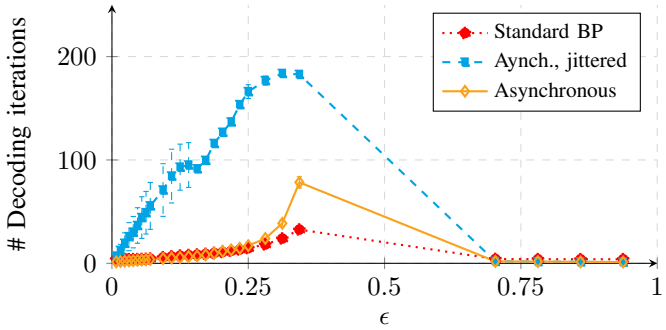


Fig. 5: Number of decoding iterations for the three different decoders

the standard BP, specially close to and beyond the Shannon's capacity limit ($\epsilon^* = 0.5$ in this case). As mentioned earlier in the paper, the good performance of standard (synchronous) BP with respect the number of decoding iterations is partly because of its efficient quitting strategy (i.e., it is know when it can correct erasures and quickly quits when it reaches a stopping set).

Figure 6 illustrates the time of the last event processed by different decoding as a function of correctly decoded erasures. Here we see that below the BP threshold, both jittered and asynchronous decoding algorithms perform better than the standard BP. Furthermore, the asynchronous decoder performs better above BP and below Shannon's threshold.

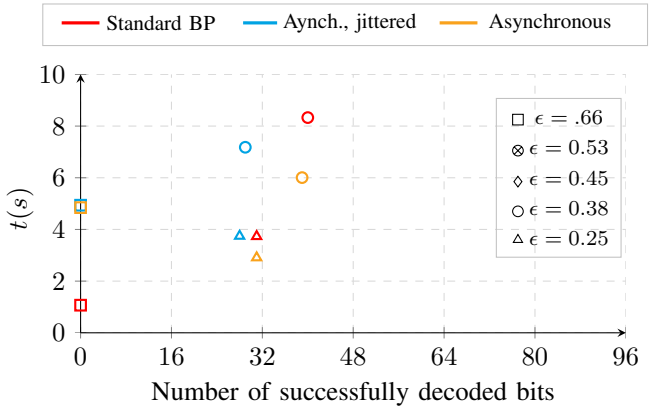


Fig. 6: Decoding time as function of the number of successfully corrected erasures, for the three different decoders.