Density Evolution for Deterministic Generalized Product Codes on the Binary Erasure Channel at High Rates

Density Evolution for Deterministic Generalized Product Codes on the Binary Erasure Channel at High Rates

Christian Häger, *Student Member, IEEE*, Henry D. Pfister, *Senior Member, IEEE*, Alexandre Graell i Amat, *Senior Member, IEEE*, and Fredrik Brännström, *Member, IEEE*

***Abstract*—Generalized product codes (GPCs) are extensions of product codes (PCs) where code symbols are protected by two component codes but not necessarily arranged in a rectangular array. We consider a deterministic construction of GPCs (as opposed to randomized code ensembles) and analyze the asymptotic performance over the binary erasure channel under iterative decoding. Our code construction encompasses several classes of GPCs previously proposed in the literature, such as irregular PCs, block-wise braided codes, and staircase codes. It is assumed that the component codes can correct a fixed number of erasures and that the length of each component code tends to infinity. We show that this setup is equivalent to studying the behavior of a peeling algorithm applied to a sparse inhomogeneous random graph. Using a convergence result for these graphs, we derive the density evolution equations that characterize the asymptotic decoding performance. As an application, we discuss the design of irregular GPCs employing a mixture of component codes with different erasure-correcting capabilities.**

***Index Terms*—Binary erasure channel, braided codes, density evolution, generalized low-density parity-check codes, inhomogeneous random graphs, multi-type branching processes, product codes, staircase codes.**

I. Introduction

Many code constructions are based on the idea of building longer codes from shorter ones [1]–[3]. In particular, product codes (PCs), originally introduced by Elias in 1954 [4], are constructed from two linear component codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, with respective lengths n_1 and n_2. The codewords in a PC are rectangular $n_1 \times n_2$ arrays such that every row is a codeword in $\mathcal{C}_1$ and every column is a codeword in $\mathcal{C}_2$. In 1981, Tanner significantly extended this construction and introduced generalized low-density parity-check (GLDPC) codes [5]. GLDPC codes are defined via bipartite graphs where variable nodes (VNs) and constraint nodes (CNs) represent code symbols and component code constraints, respectively. If the underlying graph of a GLDPC code consists exclusively of degree-2 VNs (i.e., each code symbol is protected by two component codes), the code is referred to as a generalized PC (GPC). Most of the examples presented in [5] fall into this category.

PCs have an intuitive iterative decoding algorithm and are used in a variety of applications [6], [7]. In practice, the component codes are typically Bose–Chaudhuri–Hocquenghem (BCH) or Reed–Solomon codes, which can be efficiently decoded via algebraic bounded-distance decoding (BDD). This makes GPCs particularly suited for high-speed applications due to their significantly reduced decoding complexity compared to message-passing decoding of low-density parity-check (LDPC) codes [8]. For example, GPCs have been investigated by many authors as practical solutions for forward-error correction in fiber-optical communication systems [8]–[15].

The iterative decoding of GPCs is a standard element in many of these systems and the analysis of iterative decoding is typically based on density evolution (DE) [16], [17] using an ensemble argument. That is, rather than analyzing a particular code directly, one considers a set of codes, defined via suitable randomized connections between VNs and CNs in the Tanner graph. Some notable exceptions include Gallager's original analysis based on deterministic constructions of large-girth LDPC codes [18], Tanner's analysis of Hamming GPCs [5], the analysis of PCs using monotone graph properties [19], and the analysis of PCs based on the k-core problem [9], [11].

In this paper, we focus on the asymptotic performance of GPCs over the binary erasure channel (BEC) assuming iterative decoding based on BDD of the component codes. In particular, we consider the case where the component codes have a fixed erasure-correcting capability and the length of each component code tends to infinity. Like [9], [11], [19], we consider a *deterministic* construction of GPCs. Indeed, many classes of GPCs have a very regular structure in terms of their Tanner graph and are not at all random-like. The code construction we consider is sufficiently general to recover several of these classes as special cases, such as irregular PCs [20], [21], block-wise braided codes [22, Sec. III], and staircase codes [8]. The main contribution of this paper is to show that, analogous to DE for code ensembles, the asymptotic performance of the considered GPC construction is rigorously characterized by a recursive update equation.

Like [9], [11], [19], this paper is largely based on results

This work was partially funded by the Swedish Research Council under grant #2011-5961. Parts of this paper were presented at the *Optical Fiber Communication Conference, Anaheim, CA, 2016*, the *International Symposium on Turbo Codes and Iterative Information Processing, Brest, France, 2016*, and the *IEEE International Symposium on Information Theory, Barcelona, Spain, 2016*.

This work was conducted when C. Häger was with the Department of Signals and Systems, Chalmers University of Technology, SE-41296 Gothenburg, Sweden. He is now with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: ch303@duke.edu).

H. Pfister is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: henry.pfister@duke.edu).

A. Graell i Amat and F. Brännström are with the Department of Signals and Systems, Chalmers University of Technology, SE-41296 Gothenburg, Sweden (e-mail: {alexandre.graell, fredrik.brannstrom}@chalmers.se).

that have been derived in random graph theory. In our case, the Tanner graph itself is deterministic and consists of a fixed arrangement of (degree-2) VNs and CNs. Randomness is introduced entirely due to the channel by forming the so-called residual graph (or error graph) from the Tanner graph, i.e., after removing known VNs and collapsing erased VNs into edges [9], [11], [19]. Thus, different channel realizations give rise to an ensemble of residual graphs, facilitating the analysis. The code construction considered here is such that the residual graph ensemble corresponds to the sparse inhomogeneous random graph model in [23]. Analyzing the decoding failure of the iterative decoder (for a fixed number of iterations) can then be translated into a graph-theoretic question about the behavior of a peeling algorithm applied to such a random graph. We can then use a convergence result in [23] to conclude that, as the number of vertices in the graph tends to infinity, the correct limiting behavior is obtained by evaluating the peeling algorithm on a multi-type branching process.

A similar connection between large random graphs and branching processes also arises in the DE analysis for code ensembles, e.g., irregular LDPC codes. The main difference between this and our setup is that, for code ensembles, the Tanner graph itself is random due to the randomized edge connections in the ensemble definition. DE relies on the fact that the asymptotic behavior of an extrinsic iterative message-passing decoder can be analyzed by considering an ensemble of computation trees [24, Sec. 3.7.2] (see also [25, Sec. 1]). This tree ensemble can alternatively be viewed as a multi-type branching process, where types correspond to VNs and CNs of different degrees. A tree-convergence and concentration result ensures that the performance of a code taken (uniformly at random) from the ensemble will be close to the predicted DE behavior, provided that the code is sufficiently long [17, Th. 2].

The above ensemble approach can be applied to GLDPC codes and thus also to GPCs. For example, in [26]–[28] a DE analysis for protograph-based braided codes is presented, where the Tanner graph of a tightly-braided code is interpreted as a protograph [29]. An ensemble approach has been further applied to regular GPCs in [30], where the authors analyze the asymptotic ensemble performance and derive the corresponding iterative decoding thresholds. In [31]–[33], the authors perform a DE analysis for GPC ensembles paying special attention to so-called spatially-coupled codes. On the other hand, many GPCs proposed for practical systems (e.g., the recent code proposals for optical transport networks in [8] and [13]) are entirely deterministic and not based on a randomized code ensemble. One reason for this is that deterministic GPCs have been shown to achieve extremely low error floors in practice. Moreover, the inherent code structure often results in implementation advantages compared to randomized GPCs. For example, the array representation of many deterministic GPCs facilitates the use of simple hardware layouts and efficient "row-column" iterative decoding schedules, whereas ensemble-based GPCs are unlikely to possess an array representation. Therefore, given the structured Tanner graphs of many practical GPC classes, it would be highly desirable to make precise statements about the performance of actual codes, without resorting to an ensemble argument.

The work here is closely related to [9], [11], [19]. In [19], combinatorial tools from the study of random graphs are used to analyze the iterative decoding of PCs. In [9], the authors point out the direct connection between the iterative decoding of PCs and a well-studied problem in random graph theory: the emergence of a k–core, defined as the largest induced subgraph where all vertices have degree at least k [34]. Indeed, assuming that all component codes can correct $\mathtt{t}$ erasures and allowing for an unrestricted number of iterations, the decoding either finishes successfully, or gets stuck and the resulting graph corresponds to the $(\mathtt{t}+1)$–core of the residual graph. The results in [34] apply to PCs only after some modifications (described in [9]), since the random graph model in [34] is slightly different than the actual one corresponding to the residual graph ensemble of PCs. In a later paper, Justesen considered GPCs for which the Tanner graph is based on a complete graph [11] (see, e.g., Fig. 1(b)). In that case, the results in [34] are directly applicable. The resulting codes are referred to as half-product codes (HPCs). Even though these codes have received very little attention in the literature, Tanner already used a similar construction [5, Fig. 6].

We use HPCs as the starting point for our analysis. The reason is that the residual graph of an HPC corresponds exactly to an instance of the Erdős–Rényi random graph model $\mathcal{G}(n,p)$ [35], [36], which is arguably one of the most well-studied random graph models and also considerably simpler than the inhomogeneous random graph model in [23]. It is therefore instructive to consider this case in sufficient detail before analyzing generalizations to other GPCs. Even though other classes of GPCs are mentioned and discussed also in [11] (e.g., braided codes), so far, rigorous analytical results about the asymptotic performance of deterministic GPCs have been limited to conventional PCs and HPCs.

As an application of the derived DE equations for deterministic GPCs, we discuss the optimization of component code mixtures for HPCs. In particular, we consider the case where the component codes can have different erasure-correcting capabilities. It is shown that, similar to irregular PCs [20], [21], HPCs greatly benefit from employing component codes with different strengths, both in terms of decoding thresholds and finite-length performance. We further derive upper and lower bounds on the iterative decoding thresholds of HPCs with component code mixtures. The upper bound is shown to have a graphical interpretation in terms of areas related to the DE equations, similar to the area theorem of irregular LDPC codes.

The remainder of the paper is structured as follows. We start by analyzing HPCs in Sections II, III, and IV. In particular, in Section II we discuss the code construction, the decoding algorithm, and state the main result about the asymptotic performance of HPCs in Theorem 1. In Section III, we review the necessary background about random graphs and branching processes related to the proof of Theorem 1, which is then given in Section IV. In Section V, we extend Theorem 1 to a general deterministic construction of GPCs and derive the corresponding DE equations. The optimization of component code mixtures for irregular HPCs is studied in Section VII. The paper is concluded in Section VIII.

A. Notation

The following notation is used throughout the paper. We define the sets $[n] \triangleq \{1, 2, \dots, n\}$, $\mathbb{N}_0 \triangleq \{0, 1, 2, \dots\}$, and $\mathbb{N} \triangleq \{1, 2, \dots\}$. The cardinality of a set $\mathcal{A}$ is denoted by $|\mathcal{A}|$. Sequences are denoted by $(x_n)_{n\geq 1} = x_1, x_2, \dots$. The probability density function (PDF) of a random variable (RV) X is denoted by $f_X(\cdot)$. Expectation and probability are denoted by $\mathbb{E}[\cdot]$ and $\mathbb{P}(\cdot)$, respectively. We write $X \sim \mathsf{B}(p)$ if X is a Bernoulli RV with success probability p, $X \sim \mathsf{Bin}(n, p)$ if X is a Binomial RV with parameters n and p, and $X \sim \mathsf{Po}(\lambda)$ if X is a Poisson RV with mean λ. With some abuse of notation, we write, e.g., $\mathbb{P}(\mathsf{Po}(\lambda) \geq \mathtt{t})$ for $\mathbb{P}(X \geq \mathtt{t})$ with $X \sim \mathsf{Po}(\lambda)$. We define the Poisson tail probability as $\Psi_{\geq \mathtt{t}}(\lambda) \triangleq \mathbb{P}(\mathsf{Po}(\lambda) \geq \mathtt{t}) = 1 - \sum_{i=0}^{\mathtt{t}-1} \Psi_{=i}(\lambda)$, where $\Psi_{=i}(\lambda) \triangleq \frac{\lambda^i}{i!} e^{-\lambda}$. We use boldface to denote vectors and matrices (e.g., $\boldsymbol{a}$ and $\mathbf{A}$). Matrix transpose is denoted by $(\cdot)^{\mathsf{T}}$. Convergence in distribution (weak convergence) is denoted by $\xrightarrow{\mathrm{d}}$ and convergence in probability by $\xrightarrow{\mathrm{P}}$. For positive real functions, standard asymptotic notation (as $n \to \infty$) will be used, e.g., we write $f(n) = \mathcal{O}(g(n))$ if there exist constants k, n_0 such that $f(n) \leq k g(n)$ for all $n > n_0$. We write $f(n) = \Omega(g(n))$ if there exist constants k, n_0 such that $f(n) \geq k g(n)$ for all $n > n_0$. We write $f(n) = \Theta(g(n))$ if both $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$. Finally, a code is called an (n, k, d) code if it is linear and it has length n, dimension k, and minimum distance d.

II. Half-Product Codes

A. Code Construction

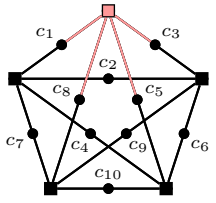
Let $\mathcal{C}$ be a binary $(n, k_{\mathcal{C}}, \mathtt{t}+1)$ code and recall that such a code can correct all erasure patterns up to weight $\mathtt{t}$. An HPC is constructed as follows (cf. [11, Sec. III-B]). Start with a conventional PC defined as the set of $n \times n$ arrays such that each row and column is a codeword in the component code $\mathcal{C}$. Then, form a subcode of this PC by retaining only symmetric codeword arrays (i.e., arrays that are equal to their transpose) with a zero diagonal. After puncturing the diagonal and the upper (or lower) triangular part of the array, one obtains an HPC of length $m = \binom{n}{2}$. The Tanner graph representing an HPC is obtained from a complete graph with n vertices by interpreting each vertex as a CN corresponding to $\mathcal{C}$ (shortened by one bit) and replacing each of the m edges by two half-edges joint together by a VN [11, Sec. III-B].[1] In the following, we assume some fixed (and arbitrary) ordering on the CNs and VNs.

Example 1. Figs. 1(a) and (b) show the code array and Tanner graph of an HPC for $n = 5$ and $m = 10$. The highlighted array elements show the code symbols participating in the second row constraint, which, due to the enforced symmetry, is also the second column constraint. Effectively, each component code acts on an L-shape in the array, i.e., both a partial row and column, which includes one diagonal element. The degree

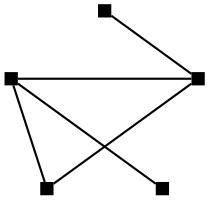
[1]One way to see this is to incorporate the symmetry constraint into the Tanner graph of a PC by connecting each VN to the "transposed" VN through a single parity-check (forcing the two to be equal). The graph now consists of degree-3 VNs (one row, one column, and one symmetry constraint), but can be simplified by removing all row (or column) constraints.

0	*	*	*	*
c_1	0	*	*	*
c_2	c_3	0	*	*
c_4	c_5	c_6	0	*
c_7	c_8	c_9	c_{10}	0

(a) 5×5 array



(b) Tanner graph



(c) residual graph

Fig. 1. Illustrations for an HPC with $n = 5$. In the array, "*" means "equal to the transposed element". The highlighted array elements illustrate one particular code constraint, which is also highlighted in the Tanner graph.

of each CN is $n - 1 = 4$, due to the zeros on the diagonal. For example, for the highlighted CN in Fig. 1(b), the second bit position of $\mathcal{C}$ is shortened (i.e., set to zero). Different bit positions are shortened for different CNs. Thus, the effective $(n-1, k_{\mathcal{C}}-1, \mathtt{t}+1)$ component codes associated with the CNs are not necessarily the same. △

Remark 1. Recall that for a Tanner graph with generalized CNs, the edges emanating from each CN should also be labeled with the corresponding component code bit positions [5, Sec. II]. For HPCs, this assignment is implicitly given due to the array description. For example, the edges emanating from the highlighted CN in Fig. 1(b) correspond to bit positions 1, 5, 4, and 3 (in left-to-right order). Reshuffling these assignments may result in an overall code with different properties (e.g., rate) even though the Tanner graph remains unchanged [5, Sec. II], [11, Sec. III-A]. However, for the considered iterative decoder, the performance remains identical as long as the component code associated with each CN is able to correct $\mathtt{t}$ erasures, regardless of the bit position assignment.

We consider the limit $n \to \infty$, i.e., we use the number of CNs in the Tanner graph to denote the problem size as opposed to the code length $m = \mathcal{O}(n^2)$. Assuming that $\mathcal{C}$ has a fixed erasure-correcting capability[2], this limit is sometimes referred to as the high-rate scaling limit or high-rate regime [31]. Indeed, if $\mathcal{C}$ has dimension $k_{\mathcal{C}}$, the rate of an HPC is lower-bounded by [7, Sec. 5.2.1] (see also [5, Th. 1])

$$R \geq 1 - \frac{n(n-1-(k_{\mathcal{C}}-1))}{m} = 1 - 2\frac{n-k_{\mathcal{C}}}{n-1}. \tag{1}$$

For a fixed erasure-correcting capability, we can assume that $n - k_{\mathcal{C}}$ in (1) stays constant. It follows that $R \to 1$ as $n \to \infty$. Note that the dimension of an HPC is $k_{\mathcal{C}}(k_{\mathcal{C}}-1)/2$ [11, Sec. III-B], [37, Lem. 8], which leads to a slightly larger rate than the lower bound in (1).

B. Binary Erasure Channel

Suppose that a codeword of an HPC is transmitted over the BEC with erasure probability p. Let I_k be the number of initial erasures associated with the k-th component code constraint. Due to symmetry, we have $\mathbb{E}[I_k] = p(n-1)$ for all $k \in [n]$. Moreover, using a Chernoff bound, it can be shown that I_k concentrates around its mean (see, e.g., [19, Sec. IV]). As a

[2]More precisely, we consider sequences of codes with increasing length and fixed erasure-correcting capability.

consequence, for a fixed $p > 0$ and $n \to \infty$, we see that any decoding attempt will be futile since $\mathbb{E}[I_k] \to \infty$ for all k, but, on the other hand, we assumed a finite erasure-correcting capability for the component codes. We therefore let the erasure probability decay slowly as $p = c/n$, for a fixed $c > 0$. Since now $p \to 0$ as $n \to \infty$, one may (falsely) conclude that decoding will always be successful in the asymptotic limit. As we will see, however, the answer depends crucially on the choice of c. It is thus instructive to interpret c as the "effective" channel quality for the chosen scaling of the erasure probability. From the above discussion, its operational meaning is given in terms of the expected number of initial erasures per component code constraint for large n, i.e., $\mathbb{E}[I_k] = c(n-1)/n \approx c$.

Remark 2. One may alternatively assume a fixed erasure probability p, in conjunction with sequences of component codes that can correct a fixed fraction of erasures in terms of their block length. However, in that case, a simple analysis reveals that the (half-)product construction is essentially useless in the limit $n \to \infty$, and it is indeed better to just use the component code by itself (see the discussion in [19, Sec. IV]).

C. Iterative Decoding

Suppose decoding is performed iteratively for ℓ iterations according to the following procedure. In each iteration, perform BDD for all CNs based on the values of the connected VNs. Afterwards, update previously erased VNs according to the decoding outcome. Updates are performed whenever there exists at least one CN where the weight of the associated erasure pattern is less than or equal to t. If the weight exceeds t, we say that the corresponding component code declares a decoding failure.

Remark 3. The decoding can alternatively be interpreted as an (intrinsic) message-passing decoder. In the first iteration, all VNs forward the received channel observations to the connected CNs. Then, CNs perform BDD based on all incoming messages and update their outgoing messages according to the decoding outcome. In subsequent iterations, outgoing VN messages are changed from erased to known if any of the two incoming CN messages becomes known. These update rules for VN and CN messages are not extrinsic (cf. [24, p. 117]), since the outgoing message along an edge may depend on the incoming message along the same edge.

An efficient way to represent the decoding is to consider the following peeling procedure. First, form the residual graph from the Tanner graph by deleting VNs and adjacent edges associated with correctly received bits and collapsing erased VNs into edges [9], [11], [19]. Then, in each iteration, determine all vertices that have degree at most t and remove them, together with all adjacent edges. The decoding is successful if the resulting graph is empty after (at most) ℓ iterations.

Example 2. Fig. 1(c) shows the residual graph for the HPC in Example 1, where c_2, c_3, c_4, c_7, and c_9 are assumed to be erased. One may check that for $\mathsf{t} = 1$, the decoding gets stuck after one iteration while for $\mathsf{t} = 2$, the decoding finishes successfully after two iterations. △

Remark 4. The above parallel peeling procedure should not be confused with the sequential "peeling decoder" described in, e.g., [24, p. 117]. That decoder uses a different scheduling where vertices are removed sequentially and not in parallel, i.e., in each step one picks only one vertex with degree at most t (uniformly at random) and removes it [24, p. 117].

D. Asymptotic Performance

For a fixed ℓ, we wish to characterize the asymptotic decoding performance as $n \to \infty$. We start by giving a heuristic argument behind the result stated in Theorem 1 below. For a similar discussion in the context of cores in random graphs, see [34, Sec. 2].

- Consider a randomly chosen CN. The decoding outcome of the BDD for this CN after ℓ iterations depends only on the depth-ℓ neighborhood[3] of the vertex in the residual graph corresponding to this CN. The residual graph itself is an instance of the Erdős–Rényi random graph model $\mathcal{G}(n,p)$, which consists of n vertices. An edge between two vertices exists with probability $p = c/n$, independently of all other edges.
- For large n, the fixed-depth neighborhood approximately looks like a Poisson branching process, which starts with an initial vertex at depth 0 that has a Poisson number of neighboring vertices with mean c that extend to depth 1. Each of these vertices has again a Poisson number of neighboring vertices, independently of all other vertices, and so on.
- For large n and fixed ℓ, one would therefore expect the probability that an individual CN declares a failure to be close to the probability that the root vertex of the first ℓ generations of the branching process survives the same peeling procedure as described for the residual graph. We define the latter probability as $z^{(\ell)}$. We will see in Section IV-C that

$$z^{(\ell)} = \Psi_{\geq \mathsf{t}+1}(cx^{(\ell-1)}), \tag{2}$$

  where the function $\Psi_{\geq \mathsf{t}}$ is defined in Section I-A and $x^{(\ell)}$ is defined recursively by $x^{(0)} = 1$ and

$$x^{(\ell)} = \Psi_{\geq \mathsf{t}}(cx^{(\ell-1)}). \tag{3}$$

The main result for HPCs is as follows.

Theorem 1. *Let W_k be the indicator RV for the event that the k-th component code declares a decoding failure after ℓ iterations of decoding and let the fraction of failed component codes be $W = \frac{1}{n}\sum_{k=1}^{n} W_k$. Then, we have*

$$\lim_{n\to\infty} \mathbb{E}[W] = z^{(\ell)}. \tag{4}$$

Furthermore, for any $\varepsilon \geq 0$, there exist $\delta > 0$, $\beta > 0$, and $n_0 \in \mathbb{N}$ such that for all $n > n_0$ we have

$$\mathbb{P}\left(|W - \mathbb{E}[W]| \geq \varepsilon\right) \leq e^{-\beta n^{\delta}}. \tag{5}$$

Proof. The proof is given in Section IV. □

[3]The depth-ℓ neighborhood of a vertex is the subgraph induced by all vertices that can be reached by taking ℓ or fewer steps from the vertex.

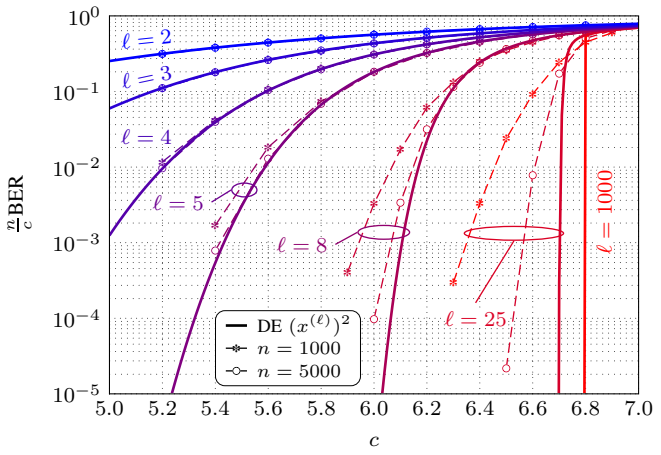


Fig. 2. DE and simulation results for HPCs with $\mathtt{t} = 4$ as a function of the iteration number ℓ.

Remark 5. In our notation, we largely suppress the dependence of the involved RVs on n and ℓ (e.g., one could write $W^{(n,\ell)}$ instead of W).

Combining (4) and (5) allows us to conclude that the code performance after ℓ iterations (measured in terms of the RV W, i.e., the fraction of component codes that declare failure) converges almost surely to a deterministic value, i.e., it sharply concentrates around $z^{(\ell)}$ for sufficiently large n. This result is analogous to the DE analysis of LDPC codes [17, Th. 2], and hence, we refer to (2) and (3) as the DE equations.

The chosen performance measure in Theorem 1 is the most natural one for the proof in Section IV. It is, however, possible to relate (2) and (3) to other performance measures that are more relevant in practice.

Example 3. The meaning of the quantity $x^{(\ell)}$ is given in Section IV-C in terms of the Poisson branching process. The operational meaning in the coding context is as follows. Consider a randomly chosen erased bit. Asymptotically, $x^{(\ell)}$ corresponds to the probablity that the bit is not recovered after ℓ decoding iterations by one of the two corresponding component codes. Since each bit is protected by two component codes, the overall probability of not recovering the bit is asymptotically given by $(x^{(\ell)})^2$. In Fig. 2, we plot the resulting DE prediction $(x^{(\ell)})^2$ as a function of c for $\mathtt{t} = 4$ and different values of ℓ, together with simulation results of the (scaled) bit error rate (BER) for $n = 1000$ and $n = 5000$. Asymptotically as $n \to \infty$, we expect the simulation results to converge to the solid lines. The rate of convergence is not analyzed in this paper, e.g., through a finite-length scaling analysis. It should be noted, however, that the convergence rate in terms of the code length m is rather "slow". More precisely, consider the gap Δ between the DE prediction and finite-length simulations for $\frac{n}{c}\text{BER} = 10^{-3}$ and $\ell = 25$ in Fig. 2. From the simulation results, one may estimate that $\Delta \approx \mathcal{O}(n^{-1/2}) = \mathcal{O}(m^{-1/4})$, since $m = \mathcal{O}(n^2)$. △

Theorem 1 can be seen as an application of [23, Th. 11.6], except for the concentration bound in (5). In fact, [23, Th. 11.6] applies to a more general class of inhomogeneous random graphs, and we use it later when studying generalizations to other GPCs. The reason for including a separate proof for HPCs in Section IV is two-fold. First, since [23, Th. 11.6] applies to a more general class of random graphs, it is instructive to consider the simplest case, i.e., the random graph $\mathcal{G}(n,p)$ corresponding to HPCs, separately and in more detail. Second, rather than relying on [23, Th. 11.6], a self-contained proof of Theorem 1 allows us to point out similarities and differences to the DE analysis for LDPC codes in [17], [25], which we believe many readers are familiar with.

As mentioned in [11], iterative decoding of HPCs over the BEC is closely related to the emergence of a k–core in $\mathcal{G}(n,p)$. First observe that the overall decoding is successful if the RV W is strictly zero, i.e., if none of the component decoders declare failure. The existence of a core can then be related to the overall decoding failure assuming an unrestricted number of iterations. Therefore, there is a subtle difference between studying the core and the overall decoding failure in our setup. In our case, the notion of decoding failure is always linked to the number of decoding iterations, which is assumed to be fixed (cf. [24, Sec. 3.19]). As a consequence, even though the overall decoding may fail after a finite number of iterations, there need not be a core in the residual graph. (The decoding may have been successful if we had done one more iteration, say.) Linking the decoding failure to the number of iterations has the advantage that it can always be determined locally (within the neighborhood of each vertex), whereas the core is a global graph property. In general, additional effort is required to infer information about global graph properties from local ones [38, Sec. 3.3], [39].

E. Performance Prediction of Finite-Length Codes

Before proving Theorem 1, it is instructive to discuss the practical implications of the asymptotic DE analysis for finite-length codes. This is particularly important because the high-rate scaling limit implies $p \to 0$ and $R \to 1$ as $n \to \infty$, which seems to preclude any practical usefulness. To see that this is not the case, we start by reviewing the practical usefulness of DE for finite-length LDPC codes.

DE is typically used to find decoding thresholds that divide the channel quality parameters range (e.g., the erasure probability or the signal-to-noise ratio) into a region where reliable communication is possible and where it is not. This interpretation of the threshold as a sharp dividing line is appropriate for $n \to \infty$, where, for LDPC codes, n is the code length. On the other hand, for finite n, thresholds are still useful to approximately predict the region of the channel quality parameter range where the performance curve of a finite-length LDPC code bends into the characteristic waterfall behavior. Moreover, thresholds have been used with great success as an optimization criterion to improve the performance of practical, finite-length LDPC codes in a wide variety of applications. The rationale behind this approach is that threshold improvements translate quite well into performance improvements, at least if n is sufficiently large. While there is no guarantee that this approach works, it typically leads to fast and efficient optimization routines.

The asymptotic DE analysis in this paper can be used in essentially the same way. The main conceptual difference with respect to DE for LDPC codes is that decoding thresholds are not given in terms of the actual channel quality, but rather in terms of the effective channel quality. In particular, the decoding threshold is formally defined as

$$c^* \triangleq \sup\{c > 0 \,|\, \lim_{\ell\to\infty} z^{(\ell)} = 0\}. \tag{6}$$

Asymptotic results, including thresholds, can be translated into a nonasymptotic setting by considering the erasure probability scaling $p = c/n$ for a given (finite) n. For example, consider the results for HPCs with $\mathtt{t} = 4$ shown in Fig. 2. The threshold in this case is located at approximately $c^* \approx 6.8$. Therefore, we should expect the waterfall behavior for $n = 1000$ to start at $p \approx 0.0068$ and for $n = 5000$ at $p \approx 0.00136$.

The practical usefulness of the asymptotic DE analysis for finite-length codes will be further illustrated in Section VII, where we consider the parameter optimization of so-called irregular HPCs. We will see that by using thresholds as an optimization criterion, performance improvements for finite-length codes can be obtained in much the same way as for LDPC codes.

III. Random Graphs and Branching Processes

In this section, we review the necessary background related to the proof of Theorem 1 in Section IV.

A. Random Graphs

Let $\mathcal{G}(n,p)$ be the Erdős–Rényi model (also known as the Gilbert model) of a random graph with n vertices, where each of the $m = \binom{n}{2}$ possible edges appears with probability p, independently of all other edges [35], [36]. A helpful representation of this model is to consider a random, symmetric $n \times n$ adjacency matrix $\boldsymbol{\theta}$ with entries $\theta_{i,i} = 0$ and $\theta_{i,j} (= \theta_{j,i}) \sim \mathsf{B}(p)$. We use G to denote a random graph drawn from $\mathcal{G}(n,p)$. For the remainder of the paper, we fix $p = c/n$.

Example 4. Let $D_k = \sum_{j=1}^{n} \theta_{k,j}$ be the degree of the k-th vertex. For any $k \in [n]$, $D_k \sim \mathsf{Bin}(n-1, c/n)$ with $\mathbb{E}[D_k] = (n-1)c/n$. For large n, all degrees are approximately Poisson distributed with mean c. More precisely, let $(D_n)_{n\geq 1}$ be a sequence of RVs denoting the degrees of randomly chosen vertices in $\mathcal{G}(n, c/n)$ and $D \sim \mathsf{Po}(c)$. Then, $D_n \xrightarrow{\mathrm{d}} D$. △

The following result about the maximum vertex degree will be used in the proof of the concentration bound (5).

Lemma 1. *Let $D_{\max} \triangleq \max_{i\in[n]} \sum_{j=1}^{n} \theta_{i,j}$ be the maximum degree of all vertices in the random graph G. We have*

$$\mathbb{P}(D_{\max} \geq d_n) \leq e^{-\Omega(d_n)}, \tag{7}$$

where d_n is any function of n satisfying $d_n = \Omega(\log(n))$.

Proof. The proof is standard and relies on Chernoff's inequality and the union bound. For completeness, a proof is given in Appendix A. □

The random graph G is completely specified by all its edges, i.e., by the m RVs $\theta_{i,j}$ for $1 \leq j < i \leq n$. It is sometimes more convenient to specify these RVs in a length-m vector instead of a matrix. With some abuse of notation, we also write $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)^{\mathsf{T}}$, asserting that there is a one-to-one correspondence between θ_k and $\theta_{i,j}$.

Example 5. Let $E = \sum_{k=1}^{m} \theta_k$ be the number of edges in G. Then, $E \sim \mathsf{Bin}(m, c/n)$ and the expected number of edges grows linearly with n since $\mathbb{E}[E] = mp = (n-1)c/2$. △

B. Neighborhood Exploration Process

An important tool to study the neighborhood of a vertex in $\mathcal{G}(n,p)$ is the so-called exploration process which we briefly review in the following (see, e.g., [40, Sec. 10.4], [41, Ch. 4] for details). This process explores the neighborhood in a breadth-first manner, exposing one vertex at a time. Since we are only interested in exploring the neighborhood up to a fixed depth, we modify the exploration compared to [40, Sec. 10.4], [41, Ch. 4] and stop the process once all vertices in the entire neighborhood for a given depth ℓ are exposed. During the exploration, a vertex can either be active, explored, or neutral. At the beginning (time $t = 0$), one vertex v is active and the remaining $n-1$ vertices are neutral. At each time $t \geq 1$, we repeat the following steps.

1) Choose any of the active vertices that are closest (in terms of graph distance) to v and denote it by w. At time $t = 1$, choose v itself.
2) Explore all edges (w, w'), where w' *runs through all active vertices*. If such an edge exists, the explored neighborhood is not a tree. (Apart from this fact, this step has no consequences for the exploration process.)
3) Explore all edges (w, w'), where w' *runs through all neutral vertices*. Set w' active if the edge exists.
4) Set w explored.

Let X_t be the number of vertices that become active at time t (i.e., in step 3). The number of active vertices, A_t, and neutral vertices, N_t, at the end of time t is given by

$$A_t = A_{t-1} + X_t - 1, \quad N_t = n - t - A_t, \tag{8}$$

with $A_0 = 1$. One can also explicitly write

$$A_t = S_t - (t-1), \quad N_t = (n-1) - S_t, \tag{9}$$

where $S_t \triangleq \sum_{i=1}^{t} X_i$. Given N_{t-1}, we have that $X_t \sim \mathsf{Bin}(N_{t-1}, p)$ because each neutral vertex can become active at time t with probability p [40, p.165].

We define the stopping time J_ℓ of the process $(X_t)_{t\geq 1}$ to be the time when the entire depth-ℓ neighborhood has been exposed.[4] Formally, J_ℓ is recursively defined as

$$J_\ell = \sum_{i=1}^{J_{\ell-1}} X_i + 1 = S_{J_{\ell-1}} + 1, \tag{10}$$

for $\ell \in \mathbb{N}$, where $J_0 = 0$ (i.e., $J_1 = 1$, $J_2 = X_1 + 1$, $J_3 = \sum_{i=1}^{X_1+1} X_i + 1$, and so on).

[4] In [40, Sec. 10.4], [41, Ch. 4], the exploration process is used to study the connected components in $\mathcal{G}(n,p)$. In that case, the stopping time is commonly defined as the hitting time $J \triangleq \inf\{t \in \mathbb{N} : A_t = 0\}$, i.e., the time when we run out of active vertices during the exploration.

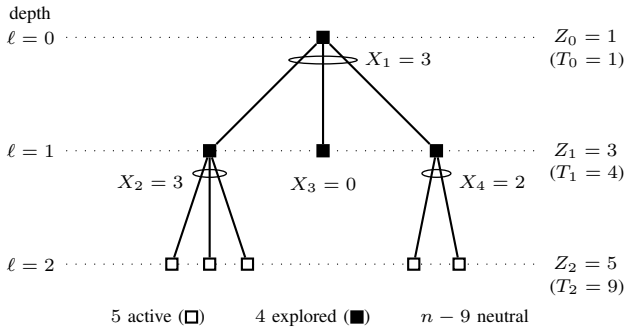


Fig. 3. The neighborhood of depth $\ell = 2$ after $J_2 = 4$ steps in the exploration corresponding to Example 6 in the text.

We further use Z_ℓ to denote the number of vertices at depth ℓ, where $Z_0 = 1$, and we let $T_\ell = \sum_{l=0}^{\ell} Z_l$ be the total number of vertices in the entire depth-ℓ neighborhood. Observe that $Z_\ell = A_{J_\ell}$, i.e., the number of vertices at depth ℓ corresponds to the number of active vertices at the stopping time J_ℓ. We also have $J_\ell = T_{\ell-1}$, i.e., the stopping time for depth ℓ corresponds to the number of all vertices up to depth $\ell - 1$.

Example 6. Assume $\ell = 2$. An example of the neighborhood is shown in Fig. 3. The corresponding realization of the stopped exploration process $(X_1, \ldots, X_{J_\ell})$ is given by $(3, 3, 0, 2)$, where we assumed a left-to-right ordering of vertices. We have $J_2 = T_1 = 4$. Observe that all vertices in the neighborhood are exposed. However, there may still be connections between any of the (active) vertices at depth 2, in which case the neighborhood contains cycles. △

C. Branching Processes

A (Galton–Watson) branching process with offspring distribution $\bar{\xi}$ is a discrete-time Markov chain $(\bar{Z}_\ell)_{\ell \geq 0}$ defined by [42, Ch. 8]

$$\bar{Z}_0 = 1 \quad \text{and} \quad \bar{Z}_{\ell+1} = \sum_{i=1}^{\bar{Z}_\ell} \bar{\xi}_{\ell,i}, \tag{11}$$

where $(\bar{\xi}_{\ell,i})_{\ell,i \geq 0}$ is a two-dimensional sequence of independent and identically distributed (i.i.d.) $\mathbb{N}_0$-valued RVs with distribution $\bar{\xi}_{\ell,i} \sim \bar{\xi}$. In our context, the interpretation of the process is as follows. Start with one vertex at depth $\ell = 0$ which has a random number of neighboring (or offspring) vertices extending to depth 1. Each of the vertices at depth 1 (if there are any) has again a random number of offspring vertices, independently of all other vertices, and so on. $\bar{Z}_\ell$ is the total number of vertices at depth ℓ, whereas $\bar{\xi}_{\ell,i}$ is the number of offspring vertices of the ith vertex at depth ℓ. We further define the total number of vertices up to (and including) depth ℓ as $\bar{T}_\ell = \sum_{l=0}^{\ell} \bar{Z}_l$.

The exploration process in the previous subsection is closely related to a Poisson branching process with mean c, i.e., the case where $\bar{\xi} = \mathsf{Po}(c)$. The connection becomes apparent by considering the random-walk perspective of the branching process [41, Sec. 3.3]. Here, the number of offspring vertices is specified in a one-dimensional fashion, indexed by t, and denoted by $\bar{X}_t$. The indexing is done breadth-first, in a predetermined order, e.g., left to right. In particular, we have

$$\bar{A}_t = \bar{A}_{t-1} + \bar{X}_t - 1 \tag{12}$$

with $\bar{A}_0 = 1$, similar to (8). The crucial difference with respect to the exploration process is that $\bar{X}_t \sim \bar{\xi}$ for all t.

Similar to the exploration process, we recursively define the stopping time for the process $(\bar{X}_t)_{t \geq 1}$ as $\bar{J}_\ell = \sum_{i=1}^{\bar{J}_{\ell-1}} \bar{X}_i + 1$ with $\bar{J}_0 = 0$ (cf. (10)), where $\bar{J}_\ell = \bar{T}_{\ell-1}$. Thus, the stopped process $(\bar{X}_1, \cdots, \bar{X}_{\bar{J}_\ell})$ specifies the branching process up to depth ℓ.

IV. Proof of Theorem 1

In the following, we provide a proof of Theorem 1. In Section IV-A, we show that, with high probability, the depth-ℓ neighborhood of a vertex in the residual graph G is a tree. We use this result in Section IV-B to show the convergence of the expected decoding outcome for an individual CN after ℓ iterations to the decoding outcome when evaluated on the branching process. The iterative decoding on the branching process (also known as DE) is analyzed in Section IV-C. Finally, the concentration bound in (5) is shown in Section IV-D.

The tree-like behavior and the convergence of the neighborhood in $\mathcal{G}(n, c/n)$ to the Poisson branching process are certainly well-known within the random-graph-theory literature. For example, this type of convergence is sometimes referred to as local weak convergence, see, e.g., [43] or [44, Prop. 2.3.1]. Here, we give a simple proof based on stochastic processes and stopping times.

A. Tree-like Neighborhood

Lemma 2. *Let $B_G(k, \ell)$ denote the depth-ℓ neighborhood of the k-th vertex in G. Then, for any $k \in [n]$, we have*

$$\mathbb{P}\left(B_G(k, \ell) \text{ is a tree}\right) \geq 1 - \frac{\beta(c, \ell)}{n}, \tag{13}$$

where $\beta(c, \ell)$ depends only on c and ℓ.

Proof. We can use the exploration process in Section III-B to show that the total number of potential edges that could create a cycle during the exploration (i.e., in step 2) is given by

$$N_\ell = \sum_{i=1}^{J_\ell} (A_{i-1} - 1) + \binom{Z_\ell}{2}. \tag{14}$$

In particular, at each time t, one vertex out of the A_{t-1} active vertices is being explored. The other $A_{t-1} - 1$ active vertices are known to be part of the neighborhood. Hence, any of the $A_{t-1} - 1$ potential edges to these vertices would create a cycle. The sum in (14) counts all of these potential edges up to the random stopping time J_ℓ. Furthermore, at the stopping time J_ℓ, there exist Z_ℓ active vertices at depth ℓ, with $\binom{Z_\ell}{2}$ potential edges between them, each of which creates a cycle (see, e.g., Fig. 3). For the neighborhood to be a tree, all of these edges must be absent. Since any edge in the exploration will not

appear with probability $1-c/n$, independently of all other edges, we have

$$\mathbb{P}\left(B_G(k,\ell) \text{ is a tree}\right) = \mathbb{E}\left[\left(1-\frac{c}{n}\right)^{N_\ell}\right] \tag{15}$$

$$\geq 1-\frac{c}{n}\mathbb{E}\left[N_\ell\right]. \tag{16}$$

Surely, N_ℓ cannot be larger than the total number of possible edges in the neighborhood, i.e., $N_\ell \leq \binom{T_\ell}{2} \leq T_\ell^2/2$, where we recall that T_ℓ is the total number of vertices encountered. Inserting this bound into (16) and using the bound (107) on $\mathbb{E}[T_\ell^2]$ in Appendix B (which depends only on c and ℓ) completes the proof. □

Remark 6. The analogous result for (regular) LDPC code ensembles is given in [17, App. A] (see also [45, Sec. 2.2]). The main difference with respect to the proof in [17, App. A] (and its extension to irregular ensembles with bounded maximum VN and CN degree) is that the number of vertices in the neighborhood cannot be upper bounded by a constant which is independent of n. (In [17], n corresponds to the LDPC code length.)

B. Convergence to the Poisson Branching Process

It is well-known that the degree of a vertex in $\mathcal{G}(n,c/n)$ converges to a Poisson RV with mean c as $n\to\infty$ (see Example 4). More generally, for any finite t, and any $(x_1,\cdots,x_t)\in\mathbb{N}_0^t$, one can easily show that (see, e.g., [41, Sec. 4.1.2])

$$\lim_{n\to\infty} f_{X_1,\ldots,X_t}(x_1,\ldots,x_t) = f_{\bar{X}_1}(x_1)\cdot\ldots\cdot f_{\bar{X}_t}(x_t), \tag{17}$$

where $\bar{X}_1,\ldots,\bar{X}_t$ are i.i.d. $\mathsf{Po}(c)$. This, together with Lemma 2, implies that the distribution on the shape of the neighborhood (for any fixed depth) converges to a Poisson branching process with mean c. To see this, note that under the assumption that the neighborhood is tree-like, its shape is specified by the stopped exploration process $(X_1,\ldots,X_{J_\ell})$. Each realization of $(X_1,\cdots,X_{J_\ell})$ is a vector of some (finite) length specifying the number of offspring vertices in the tree in a sequential manner. The set of all realizations is thus a subset of $\mathbb{N}_0^*=\mathbb{N}_0\cup\mathbb{N}_0^2\cup\mathbb{N}_0^3\cup\cdots$. Since $\mathbb{N}_0^*$ is countably infinite, there exists a one-to-one mapping between $\mathbb{N}_0^*$ and $\mathbb{N}_0$. We denote such a mapping by $\mathcal{M}:\mathbb{N}_0^*\to\mathbb{N}_0$ and let $\mathcal{M}^{-1}$ be its inverse. We now define new RVs $B_n=\mathcal{M}(X_1,\cdots,X_{J_\ell})$ and $B=\mathcal{M}(\bar{X}_1,\cdots,\bar{X}_{\bar{J}_\ell})$. One can think about enumerating all possible trees and assigning an index to each of them. A distribution over the shape of the trees is then equivalent to a distribution over the indices. It is now easy to show that $B_n\xrightarrow{\mathrm{d}}B$. For any $b\in\mathbb{N}_0$, there exists some t such that $\mathcal{M}^{-1}(b)=(x_1,\cdots,x_t)\in\mathbb{N}_0^t$. Therefore, we have

$$\lim_{n\to\infty}\mathbb{P}\left(B_n=b\right) \tag{18}$$

$$=\lim_{n\to\infty} f_{J_\ell|X_1,\ldots,X_t}(t|x_1,\cdots,x_t) f_{X_1,\ldots,X_t}(x_1,\cdots,x_t) \tag{19}$$

$$= f_{\bar{J}_\ell|\bar{X}_1,\ldots,\bar{X}_t}(t|x_1,\cdots,x_t)\lim_{n\to\infty} f_{X_1,\ldots,X_t}(x_1,\cdots,x_t) \tag{20}$$

$$\overset{(17)}{=} f_{\bar{J}_\ell|\bar{X}_1,\ldots,\bar{X}_t}(t|x_1,\cdots,x_t) f_{\bar{X}_1}(x_1)\cdot\ldots\cdot f_{\bar{X}_t}(x_t) \tag{21}$$

$$=\mathbb{P}\left(B=b\right), \tag{22}$$

where, to obtain (20) from (19), we used the fact that the conditional distributions of the stopping times J_ℓ and $\bar{J}_\ell$ given $X_1,\ldots,X_t$ and $\bar{X}_1,\ldots,\bar{X}_t$, respectively, are both independent of n and equal to 1. This is because the realizations $x_1,\ldots,x_t$ fully determine the stopping times as $J_\ell=\bar{J}_\ell=t$.

Remark 7. In general, the distribution $f_{J_\ell|X_1,\ldots,X_t}$ is not independent of n. From (10), recall that $J_\ell=\sum_{i=1}^{J_{\ell-1}}X_i+1$. One may distinguish two cases: In the first case, the realizations of $X_1,\ldots,X_t$ determine J_ℓ and $f_{J_\ell|X_1,\ldots,X_t}$ is an indicator function that does not depend on n. In the second case, the realizations of $X_1,\ldots,X_t$ do not determine J_ℓ and $f_{J_\ell|X_1,\ldots,X_t}$ depends on n. For example, let $(X_1,X_2,X_3)=(2,2,3)$. This determines $J_3=8$, i.e., $f_{J_3|X_1,X_2,X_3}(j|2,2,3)=\mathbb{I}\{j=8\}$, where $\mathbb{I}\{\cdot\}$ is the indicator function. However, $f_{J_3|X_1,X_2,X_3}(j|3,2,3)$ depends on n, since $J_3=\sum_{i=1}^{X_1+1}X_i+1=X_4+9$. To pass from (19) to (20), we only encounter the first case. This is because the random variable B_n is defined as a function of the stopped exploration process and the corresponding realizations always determine J_ℓ.

A direct consequence of this result is that the expected value of a (bounded) function applied to the neighborhood of a vertex in $\mathcal{G}(n,c/n)$ converges to the expected value of the same function applied to the branching process. In particular, recall that the RV $W=\frac{1}{n}\sum_{k=1}^n W_k$ corresponds to the fraction of component codes that declare failures after ℓ decoding iterations. The indicator RV W_k depends only on the shape of the depth-ℓ neighborhood of the k-th vertex in the residual graph. The peeling procedure can thus be written using a function $\mathcal{D}_\ell:\mathbb{N}_0\to\{0,1\}$, such that

$$\mathbb{E}[W_k\,|\,B_G(k,\ell)\text{ is a tree}]=\mathbb{E}[\mathcal{D}_\ell(B_n)], \tag{23}$$

which, due to symmetry, is independent of k. Since $B_n\xrightarrow{\mathrm{d}}B$ and $\mathcal{D}_\ell$ is bounded, we have that [46, Sec. 10]

$$\lim_{n\to\infty}\mathbb{E}[\mathcal{D}_\ell(B_n)]=\mathbb{E}[\mathcal{D}_\ell(B)]=z^{(\ell)}, \tag{24}$$

which, together with (13), implies (4).

Remark 8. It is worth mentioning that for regular LDPC code ensembles, there is no notion of an asymptotic neighborhood distribution (in the sense of (17)) beyond the fact that cycles can be ignored. This is because the ensemble of computation trees for a CN (or VN) reduces to a single deterministic tree.

C. Density Evolution

Once the true distribution on the neighborhood-shape has been replaced by the branching process, the parameter of interest can be easily computed (cf. [34, Sec. 2], [38, p. 43]). In our case, the parameter of interest is the probability that a CN declares a decoding failure after ℓ iterations as $n\to\infty$, or, equivalently, the probability that the root vertex of the branching process survives ℓ peeling iterations. Due to the recursion that is inherent in the definition of the branching process, it is not surprising that the solution is also given in terms of a recursion. This is, of course, completely analogous to the analysis of LDPC code ensembles, see, e.g., the discussion in [45, Sec. 1]. Also, similar to LDPC codes over the BEC, we

refer to this step as DE (even though the parameter of interest does not correspond to a density).

Consider a Poisson branching process with mean c. Assume that we have a realization of this process (i.e., a tree) up to depth ℓ. We wish to determine if the root vertex survives ℓ iterations of the peeling procedure (and thus the CN corresponding to the root node declares a decoding failure). One can recursively break down the answer as follows. First, for each of the root's offspring vertices, apply $\ell-1$ peeling iterations to the subtree that has the offspring vertex as a root (and extends from depth 1 to ℓ). Then, if the number of offspring vertices that survive this peeling is less than or equal to $\mathtt{t}$, remove the root vertex. This gives the same answer as applying ℓ peeling iterations to the entire tree, since we are simply postponing the removal decision for the root to the ℓ-th iteration.

Now, in order to determine the corresponding *probability* with which the root vertex survives the peeling procedure, the crucial observation is that the root's offspring vertices are removed independently of each other, and with the same probability. This is a simple consequence of the definition of the branching process and the independence assumption between the number of offspring vertices (see Section III-C). Recall that we defined the root survival probability as $z^{(\ell)}$. Furthermore, we denote the survival probability of the root's offspring vertices by $x^{(\ell-1)}$. Initially, the number of offspring vertices is Poisson distributed with mean c. After removing each offspring vertex independently with probability $1-x^{(\ell-1)}$, the offspring distribution of the root vertex follows again a Poisson distribution, albeit with (reduced) mean $cx^{(\ell-1)}$. (This is easily seen by using characteristic functions.) Hence, we obtain (2).

Essentially the same argument can be used to determine $x^{(\ell)}$. The only difference is that for offspring vertices we have to account for the fact they are connected to the previous level with an edge. Thus, they can be removed only if less than or equal to $\mathtt{t}-1$ (and not $\mathtt{t}$) of their offspring vertices survive. This leads to the recursion (3), where the initial condition is given by $x^{(0)}=1$.

D. Concentration

The concentration bound in (5) is readily proved by using the method of typical bounded differences [47]. In particular, we can apply a special case of [47, Cor. 1.4] which is stated below (with adjusted notation) for easier referencing.

Theorem 2 ([47]). *Let $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)^\intercal$ be a vector of independent RVs with $\theta_k \sim \mathsf{B}(p)$ for all k. Let $\Gamma \subseteq \{0,1\}^m$ be an event and let $f : \{0,1\}^m \to \mathbb{R}$ be a function that satisfies the following condition. There exist Λ and Λ' with $\Lambda \leq \Lambda'$ such that whenever $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \{0,1\}^m$ differ in only one coordinate, we have*

$$|f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}')| \leq \begin{cases} \Lambda & \text{if } \boldsymbol{\theta} \in \Gamma \\ \Lambda' & \text{otherwise} \end{cases} . \tag{25}$$

Then, for any $a \geq 0$ and any choice of $\gamma \in (0,1]$, we have

$$\begin{aligned} &\mathbb{P}\left(|f(\boldsymbol{\theta}) - \mathbb{E}[f(\boldsymbol{\theta})]| \geq a\right) \leq m\gamma^{-1}\mathbb{P}\left(\boldsymbol{\theta} \notin \Gamma\right) \\ &+ \exp\left(-\frac{a^2}{2m(1-p)p(\Lambda+b)^2 + 2(\Lambda+b)a/3}\right), \end{aligned} \tag{26}$$

where $b = \gamma(\Lambda' - \Lambda)$.

In our context, $\boldsymbol{\theta}$ specifies the edges in the random graph G (see Section III). Thus, we can think about $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ as specifying two different graphs $G = G(\boldsymbol{\theta})$ and $G' = G(\boldsymbol{\theta}')$. The interpretation of the condition (25) is as follows. For any two graphs G, G' that differ in only one edge, we have $|f(G) - f(G')| \leq \Lambda'$, where f denotes a function applied to the graphs. The constant Λ' is often referred to as the Lipschitz constant [47]. The event Γ is chosen such that changing one coordinate in $\boldsymbol{\theta} \in \Gamma$ (i.e., adding or removing an edge in the graph defined by $\boldsymbol{\theta}$) changes the function by at most Λ, where Λ should be substantially smaller than Λ'. The constant Λ is referred to as the typical Lipschitz constant. In this regard, the event Γ is assumed to be a typical event, i.e., it should occur with high probability.

Remark 9. In several applications, it is possible to establish concentration bounds based solely on suitable choices for Λ'. This approach leads to the more common bounded differences inequality (also known as McDiarmid's or Hoeffding-Azuma inequality). For example, the concentration bound for LDPC code ensembles in [17, Eq. (11)] is based on this approach. However, in many cases (including the one considered here) the worst case changes corresponding to Λ' can be quite large, even though the typical changes may be small. For more details, we refer the reader to [47] and references therein.

Theorem 2 is applied as follows. We let $f(\boldsymbol{\theta}) = nW = \sum_{k=1}^{n} W_k$. Since f is the sum of n indicator RVs, we can choose $\Lambda' = n$. We further let Γ be the event that the maximum vertex degree in G, denoted by $D_{\max}$, is strictly less than n^δ for some fixed $\delta \in (0,1)$. For the typical Lipschitz constant, we choose $\Lambda = 2(\ell+1)n^{\delta\ell}$. To show that for these choices the condition (25) holds, we argue as follows. First, observe that the maximum vertex degree in both G and G' is at most n^δ since adding an edge to the graph G increases the maximum degree by at most one (and removing an edge can only decrease the maximum degree). Consider now the maximum change in $\sum_{k=1}^{n} W_k$ that can occur by adding or removing an edge between two arbitrary vertices i and j under the assumption that the maximum degree remains bounded by n^δ. Since W_k depends only on the depth-ℓ neighborhood of the k-th vertex, such a change can only affect W_k if either vertex i or j (or both) are part of the neighborhood of vertex k. But, due to the bounded maximum degree, vertex i appears in at most $\sum_{l=0}^{\ell} n^{\delta l} \leq (\ell+1)n^{\delta\ell}$ neighborhoods (and so does vertex j). Hence, the sum $\sum_{k=1}^{n} W_k$ can change by at most $2(\ell+1)n^{\delta\ell}$.

We further choose $\gamma = n^{-1}$. Since $\Lambda' = n$, this implies that $b \leq \gamma\Lambda' = 1$ and therefore we have

$$(\Lambda + b) \leq (\Lambda + b)^2 \leq 4\Lambda^2. \tag{27}$$

Consider now the second term on the right-hand side (RHS) of (26) with $a = n\varepsilon$ and $p = c/n$. We have

$$\exp\left(\frac{-(n\varepsilon)^2}{2m(1-c/n)c/n(\Lambda+b)^2 + 2(\Lambda+b)n\varepsilon/3}\right) \tag{28}$$

$$\leq \exp\left(\frac{-\varepsilon^2 n}{(8c+8\varepsilon/3)\Lambda^2}\right) = e^{-\beta_1 n^{1-2\delta\ell}} \tag{29}$$

where the inequality in (29) follows from $m \leq n^2$, $1-c/n \leq 1$ and (27), and in the last step we used $\Lambda = 2(\ell+1)n^{\delta\ell}$. Note that the implicitly defined parameter $\beta_1 > 0$ depends only on ε, c, and ℓ. In order to bound the first term on the RHS of (26), we first note that $\mathbb{P}\left(\boldsymbol{\theta} \notin \Gamma\right) = \mathbb{P}\left(D_{\max} \geq n^{\delta}\right)$. We then have

$$m\gamma^{-1}\mathbb{P}\left(\boldsymbol{\theta} \notin \Gamma\right) \leq n^3 e^{-\beta_2 n^{\delta}} \leq e^{-\beta_2 n^{\delta}/2}, \tag{30}$$

where, according to Lemma 1, the first inequality holds for some $\beta_2 > 0$ and n sufficiently large. To match the exponents in (29) and (30), we can set $\delta = (1+2\ell)^{-1}$. This proves (5) and completes the proof of Theorem 1.

Remark 10. The above proof applies to any function of the form $f = \sum_{k=1}^{n} f_k$ where f_k is an indicator function that depends only on the depth-ℓ neighborhood of the k-th vertex in G.

V. Generalized Product Codes

In this section, we analyze a deterministic construction of GPCs for which the residual graph corresponds to an inhomogeneous random graph [23]. The concept of inhomogeneity naturally arises if we wish to distinguish between different types of vertices. In our case, a type will correspond to a particular position in the Tanner graph and a certain erasure-correcting capability. HPCs can be regarded as "single-type" or homogeneous, in the sense that all CNs (and thus all vertices in the residual graph) behave essentially the same.

A. Code Construction

Our code construction is defined in terms of three parameters $\boldsymbol{\eta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\tau}$. We denote the corresponding GPC by $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$, where n denotes the total number of CNs in the underlying Tanner graph. The two parameters $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ essentially determine the graph connectivity, where $\boldsymbol{\eta}$ is a binary, symmetric $L \times L$ matrix and $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_L)^{\intercal}$ is a probability vector of length L, i.e., $\sum_{i=1}^{L} \gamma_i = 1$ and $\gamma_i \geq 0$. Since GPCs have a natural representation in terms of two-dimensional code arrays (see, e.g., Fig. 5), one may alternatively think about $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ as specifying the array shape. We will see in the following that different choices for $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ recover well-known code classes. The parameter $\boldsymbol{\tau}$ is used to specify GPCs employing component codes with different erasure-correcting capabilities and will be described in more detail at the end of this subsection.

The Tanner graph describing the GPC $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ is constructed as follows. Assume that there are L positions. Place $n_i \triangleq \gamma_i n$ CNs at each position $i \in [L]$, where we assume that n_i is an integer for all i. Then, connect each CN at position i to each CN at position j through a VN if and only if $\eta_{i,j} = 1$.

In the following, we always assume that $\eta_{i,j} = 1$ for at least one j and any $i \in [L]$ so that there are no unconnected CNs. Furthermore, we assume that the matrix $\boldsymbol{\eta}$ is irreducible, so that the Tanner graph is not composed of two (or more) disconnected graphs.

Each of the n_i CNs at position i has degree

$$d_i = \eta_{i,i}(n_i - 1) + \sum_{j \neq i} \eta_{i,j} n_j, \tag{31}$$

where the first term in (31) arises from the convention that we cannot connect a CN to itself if $\eta_{i,i} = 1$. Recall that the degree of a CN corresponds to the length of the underlying component code. Thus, all component codes at the same position have the same length. However, the component code lengths may vary across positions depending on $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$.

The total number of VNs (i.e., the length of the code) is given by

$$m = \sum_{i=1}^{L} \eta_{i,i} \binom{n_i}{2} + \sum_{1 \leq i < j \leq L} \eta_{i,j} n_i n_j \approx \frac{\boldsymbol{\gamma}^{\intercal}\boldsymbol{\eta}\boldsymbol{\gamma}}{2} n^2. \tag{32}$$

In the following, we assume some fixed (and arbitrary) ordering on the CNs and VNs.

Remark 11. In the light of Remark 1, we see that the above construction merely specifies a Tanner graph and not a code. This is due to the missing assignment of the component code bit positions to the CN edges. Since our results do not depend on this assignment, it is assumed to be (arbitrarily) fixed. In the following examples, the assignment is implicitly specified due to an array description.

Example 7. HPCs are recovered by considering $\boldsymbol{\eta} = 1$ and $\boldsymbol{\gamma} = 1$. All CNs are equivalent and correspond to component codes of length $n-1$. △

Example 8. Choosing $\boldsymbol{\eta} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ leads to a PC. The relative lengths of the row and column component codes can be adjusted through $\boldsymbol{\gamma}$, where $\boldsymbol{\gamma} = (1/2, 1/2)$ leads to a "square" PC with (uniform) component code length $n/2$. Note that the total number of CNs n is assumed to be even in this case. △

Example 9. Consider an arbitrarily-shaped code array of finite size which is composed of $n' \times n'$ blocks arranged on a grid. In Fig. 4, we illustrate how to construct $\boldsymbol{\eta}$ for such an array. This construction uses the (arbitrary) convention that column and row positions of the blocks are indexed by odd and even numbers, respectively. First, form the matrix $\boldsymbol{\eta}'$ representing the array, where entries are 1 if a block is present on the corresponding grid point and 0 otherwise. Assume that $\boldsymbol{\eta}'$ has size $a' \times b'$, and let $a = \max(a', b')$. Then, the matrix $\boldsymbol{\eta}$ is of size $2a \times 2a$ (i.e., $L = 2a$) and can be constructed by using the prescription

$$\begin{aligned} \eta_{2i,2j-1} &= \eta'_{i,j}, \\ \eta_{2i-1,2j} &= \eta'_{j,i} \end{aligned} \tag{33}$$

for $i \in [a]$ and $j \in [b]$ and $\eta_{i,j} = 0$ elsewhere. For example, consider a PC where $\boldsymbol{\eta} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\boldsymbol{\gamma} = (2/5, 3/5)$, and $n = 20$. In this case, the row and column codes have length 8 and 12, respectively. An alternative way of describing *the same* code is to assume that the code array is composed of 6 blocks of size 4×4. In this case, we obtain

$$\boldsymbol{\eta}' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\eta} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \tag{34}$$

where $\boldsymbol{\eta}'$ is the matrix describing the array and $\boldsymbol{\eta}$ results from applying (33). Note that CNs at position 5 are not connected

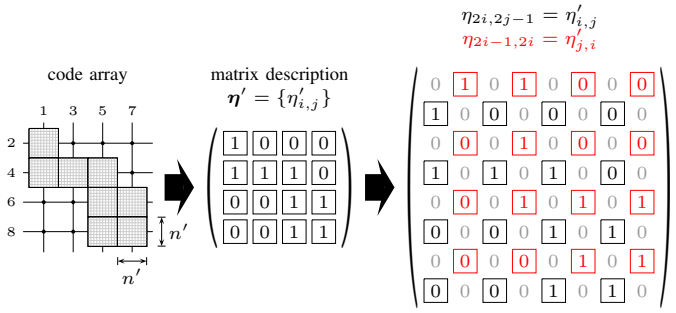
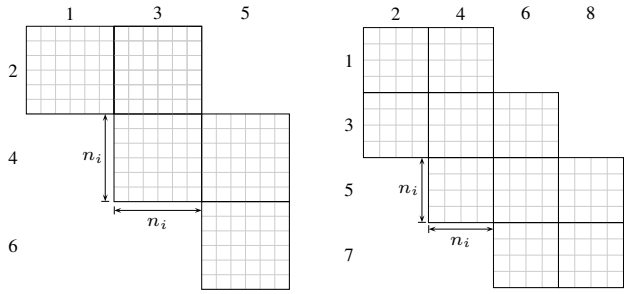


Fig. 4. Construction of $\boldsymbol{\eta}$ for an arbitrary code array composed of $n' \times n'$ blocks that are arranged on a grid. Red elements in $\boldsymbol{\eta}$ are inserted such that $\boldsymbol{\eta}$ is symmetric. With this construction, even (odd) positions in $\boldsymbol{\eta}$ correspond to row (column) codes.



(a) staircase code (b) block-wise braided code

Fig. 5. Examples of code arrays for (a) staircase codes (see Example 10) and (b) block-wise braided codes (see Example 11).

(i.e., $\eta_{5,i} = \eta_{i,5} = 0$ for all i). This is a consequence of the assumed numbering convention for column and row positions. After removing empty rows and columns from $\boldsymbol{\eta}$, we can set $L = 5$, $n = 20$, and $\gamma_i = 1/5$ for all i in order to obtain the same PC as with $\boldsymbol{\eta} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\boldsymbol{\gamma} = (2/5, 3/5)$, and $n = 20$. △

Example 10. For a fixed $L \geq 2$, the matrix $\boldsymbol{\eta}$ describing a staircase code [8] has entries $\eta_{i,i+1} = \eta_{i+1,i} = 1$ for $i \in [L-1]$ and zeros elsewhere. The distribution $\boldsymbol{\gamma}$ is uniform, i.e., $\gamma_i = 1/L$ for all $i \in [L]$. For example, the staircase code corresponding to the code array shown in Fig. 5(a), where $L = 6$ and $n = 36$ (i.e., $n_i = 6$), is defined by

$$\boldsymbol{\eta} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{35}$$

and $\gamma_i = 1/6$. The CNs at all positions have the same degree $2n\gamma_i = 12$, except for positions 1 and L, where the degrees are $n\gamma_i = 6$. △

Example 11. For even $L \geq 4$, the matrix $\boldsymbol{\eta}$ for a particular instance of a block-wise braided code has entries $\eta_{i,i+1} = \eta_{i+1,i} = 1$ for $i \in [L-1]$, $\eta_{2i-1,2i+2} = \eta_{2i+2,2i-1} = 1$ for $i \in [L/2-1]$, and zeros elsewhere. For example, we have

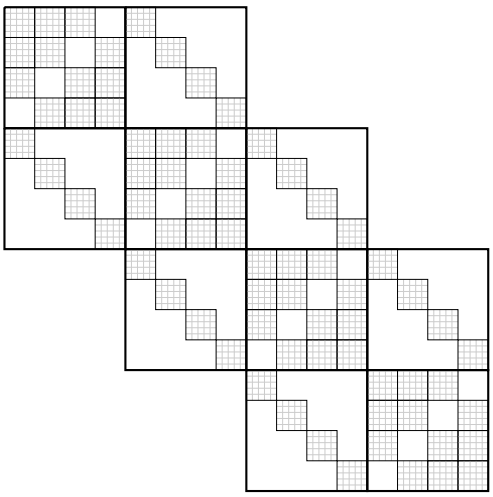


Fig. 6. A block-wise braided code where the MBPs have a block-wise structure. We have $N = 20$, $n = 160$, and the multiplicities for the diagonal and off-diagonal MBPs are 15 and 5, respectively.

$$\boldsymbol{\eta} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \tag{36}$$

for $L = 8$. The corresponding code array is shown in Fig. 5(b), where $n = 32$ and $\boldsymbol{\gamma}$ is uniform. In general, the construction of a block-wise braided code is based on so-called multiple block permutators (MBPs). An MBP with multiplicity k is an $N \times N$ matrix with k ones in each row and column [22, Def. 2.1]. Given a component code of length $n_\mathcal{C}$ and dimension $k_\mathcal{C}$, the diagonal and off-diagonal array blocks in Fig. 5(b) correspond to MBPs with respective multiplicities $2k_\mathcal{C} - n_\mathcal{C}$ and $n_\mathcal{C} - k_\mathcal{C}$, where $N \geq \min(2k_\mathcal{C} - n_\mathcal{C}, n_\mathcal{C} - k_\mathcal{C})$. However, this definition is unnecessarily narrow for our purposes in the sense that the multiplicities of the MBPs are linked to the dimension of the component code. For example, for the array shown in Fig. 5(b) (where $N = 4$ and $n = 12$), it would be required that each component code has dimension $k_\mathcal{C} = 8$ in order to comply with the definition in [22]. Here, we simply lift the constraint that the multiplicities of the MBPs are linked to the component code dimension. The only requirement for the considered GPC construction is that the MBPs have a block-wise structure themselves, see Fig. 6 for an example. Note that $\boldsymbol{\eta}$ can be found by following the steps in Example 9. △

Remark 12. Both staircase and braided codes were originally introduced as convolutional-like codes with conceptually infinite length, i.e., $L = \infty$. It then becomes customary to employ a sliding-window decoder whose analysis is discussed in Section VI-C. We also remark that it is straightforward to extend the above construction of $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ for staircase and braided codes to their natural tail-biting versions (see,

e.g., [11]). Staircase and braided codes can be classified as instances of (deterministic) spatially-coupled PCs, which are discussed in more detail in Section VI-E.

Up to this point, the GPC construction for a given $\boldsymbol{\eta}$, $\boldsymbol{\gamma}$, and n specifies the lengths of the component codes via (31). We proceed by assigning different erasure-correcting capabilities to the component codes corresponding to CNs at different positions. To that end, for $i \in [L]$, let $\boldsymbol{\tau}(i) = (\tau_1(i), \dots, \tau_{\mathsf{t}_{\max}}(i))^\intercal$ be a probability vector of length $\mathsf{t}_{\max}$, where $\tau_{\mathsf{t}}(i)$ denotes the fraction of CNs at position i (out of n_i total CNs) which can correct t erasures and $\mathsf{t}_{\max}$ is the maximum erasure-correcting capability. With some abuse of notation, the collection of these distributions for all positions is denoted by $\boldsymbol{\tau} = (\boldsymbol{\tau}(i))_{i=1}^{L}$. The assignment can be done either deterministically, assuming that $\tau_{\mathsf{t}}(i)n_i$ is an integer for all $i \in [L]$ and $\mathsf{t} \in [\mathsf{t}_{\max}]$, or independently at random according to the distribution $\boldsymbol{\tau}(i)$ for each position.

Example 12. Consider a PC where the row and column codes have the same length but different erasure-correcting capabilities t and t', respectively. We have $\boldsymbol{\eta} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\boldsymbol{\gamma} = (1/2, 1/2)$, and additionally $\tau_{\mathsf{t}}(1) = 1$ and $\tau_{\mathsf{t}'}(2) = 1$. More generally, the erasure-correcting capabilities may also vary across the row (and column) codes leading to irregular PCs [20], [21]. $\triangle$

Example 13. Staircase codes with component code mixtures were suggested (but not further investigated) in [48, Sec. 4.4.1]. The case described in [48, Sec. 4.4.1] corresponds to a fixed choice of $\boldsymbol{\tau}(i)$ which is independent of i. A code ensemble that is structurally related to staircase codes based on component code mixtures was analyzed in [33, Sec. III]. $\triangle$

B. Inhomogeneous Random Graphs

Assume that a codeword of $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ is transmitted over the BEC with erasure probability $p = c/n$, for $c > 0$. Recall that the residual graph is obtained by removing known VNs and collapsing erased VNs into edges. We now illustrate how the ensemble of residual graphs for $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ is related to the inhomogeneous random graph model.

Remark 13. The parameter c can again be interpreted as an "effective" channel quality. Unlike for HPCs, its operational meaning does not necessarily correspond to the expected number of initial erasures per component code constraint (see Section II-B). However, the construction of $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ can be slightly modified by introducing a parameter $a > 0$ that scales the total number of CNs from n to an. In this case, a can be chosen such that c corresponds again to the average number of erasures per component code constraint for large n. This modified construction is discussed in Section VI-A below.

In [23], inhomogeneous random graphs are specified by a vertex space $\mathcal{V}$ and a kernel $\boldsymbol{\kappa}$. Here, we consider only the finite-type case, see [23, Ex. 4.3]. In this case, the number of different vertex types is denoted by r and the vertex space $\mathcal{V}$ is a triple $(\mathcal{S}, \mu, (\boldsymbol{y}_n)_{n\geq 1})$, where $\mathcal{S} = [r]$ is the so-called type space, $\mu : \mathcal{S} \to [0, 1]$ is a probability measure on $\mathcal{S}$, and $\boldsymbol{y}_n = (y_1^{(n)}, y_2^{(n)}, \dots, y_n^{(n)})$ is a deterministic or random sequence of points in $\mathcal{S}$ such that for each $i \in \mathcal{S}$, we have

$$\frac{|\{k : y_k^{(n)} = i\}|}{n} \xrightarrow{\text{P}} \mu(i) \tag{37}$$

as $n \to \infty$. For a finite number of vertex types, the kernel $\boldsymbol{\kappa}$ is a symmetric $r \times r$ matrix, where entries are denoted by $\kappa_{i,j}$. For a fixed $n > \max_{i,j} \kappa_{i,j}$, the inhomogeneous random graph $\mathcal{G}^{\mathcal{V}}(n, \boldsymbol{\kappa})$ is defined as follows. The graph has n vertices where the type of vertex i is given by $y_i^{(n)}$. An edge between vertex i and j exists with probability $n^{-1}\kappa_{y_i^{(n)}, y_j^{(n)}}$, independently of all other edges.

Remark 14. Even though we use [23] as our main reference, finite-type inhomogeneous random graphs (and their relation to multi-type branching processes) were first introduced and studied in [49]. See also the discussion in [38, p. 31].

For the code $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$, the residual graph is an instance of an inhomogeneous random graph with a finite number of types, as defined above. In particular, there are $r = L\mathsf{t}_{\max}$ different types in total, i.e., we have $\mathcal{S} = [L\mathsf{t}_{\max}]$. In our case, it is more convenient to specify the type of a vertex by a pair (i, t), where $i \in [L]$ corresponds to the position in the Tanner graph and $\mathsf{t} \in [\mathsf{t}_{\max}]$ corresponds to the erasure-correcting capability. In the construction of the sequence $\boldsymbol{x}_n$, the assignment of the type corresponding to the position is always deterministic. For the type corresponding to the erasure-correcting capability, we have the freedom to do the assignment deterministically or uniformly at random. In both cases, the fraction of vertices of type (i, t) is asymptotically given by $\gamma_i \tau_{\mathsf{t}}(i)$. This specifies the probability measure μ through the condition (37). (For the random assignment, the condition (37) holds due to the weak law of large numbers.) The kernel $\boldsymbol{\kappa}$ is obtained from $\boldsymbol{\eta}$ by replacing each 0 entry with the all-zero matrix of size $\mathsf{t}_{\max} \times \mathsf{t}_{\max}$ and each 1 entry with a $\mathsf{t}_{\max} \times \mathsf{t}_{\max}$ matrix where all entries are equal to c.

Remark 15. The inhomogeneous random graph model in [23] is much more general than the finite-type case described above. In particular, $\mathcal{S}$ can be a separable metric space and κ a symmetric non-negative (Borel) measurable function on $\mathcal{S} \times \mathcal{S}$. This more general framework could be used for example to obtain the DE equations for so-called tightly-braided codes [13], [22]. However, in that case the analysis does not admit a characterization in terms of a finite number of types. In particular, the DE equations are given in terms of integrals and solving the equations may then require the application of numerical integration techniques.

Similar to the (homogeneous) random graph $\mathcal{G}(n, p)$, one may use an alternative representation in terms of a random, symmetric $n \times n$ adjacency matrix $\boldsymbol{\theta}$ with zeros on the diagonal. The structure of this matrix is shown in Fig. 7. The matrix is composed of submatrices $\boldsymbol{\theta}_{i,j}$ of size $n_i \times n_j$. The submatrix $\boldsymbol{\theta}_{i,j}$ is zero if $\eta_{i,j} = 0$ and it consists of i.i.d. $\mathsf{B}(p)$ RVs if $\eta_{i,j} = 1$ (with the constraint that the matrix $\boldsymbol{\theta}$ is symmetric and all diagonal elements are zero). The inhomogeneous random graph is thus specified by m Bernoulli RVs, where m is defined in (32).

From the matrix representation, it can be seen that the degree of a vertex at position k is distributed according

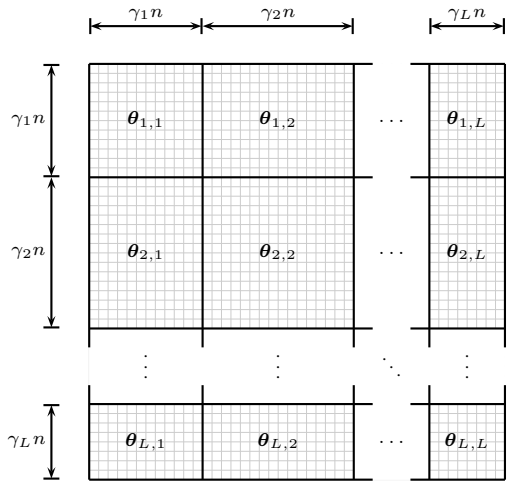


Fig. 7. The structure of the random, symmetric adjacency matrix $\boldsymbol{\theta}$.

to $\mathsf{Bin}(d_k, c/n)$, where d_k is defined in (31). Moreover, all vertices follow a Poisson distribution as $n \to \infty$, where the mean for vertices at position k is given by

$$\lim_{n\to\infty} d_k \frac{c}{n} = c \sum_{j=1}^{L} \gamma_j \eta_{k,j}. \tag{38}$$

C. Iterative Decoding

After transmission over the BEC, we apply the same iterative decoding as described in Section II-C for the HPC. The only difference is that each component code is assumed to correct all erasures up to its erasure-correcting capability. In the corresponding iterative peeling procedure for the residual graph, one removes vertices of degree at most $\mathtt{t}$, where $\mathtt{t}$ is the erasure-correcting capability of the corresponding CN. The erasure-correcting capability may now be different for different vertices depending on their type.

D. Asymptotic Performance

For a fixed number of decoding iterations ℓ, we wish to characterize the asymptotic performance as $n \to \infty$. The crucial observation is that the distribution on the neighborhood-shape of a randomly chosen vertex in the residual graph converges asymptotically to a multi-type branching process [23, Remark 2.13]. In our case, the multi-type branching process is defined in terms of the code parameters $\boldsymbol{\eta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\tau}$. It generalizes the (single-type) branching process described in Section III-C as follows. The process starts with one vertex at depth 0 which has random type $(i, \mathtt{t})$ with probability $\gamma_i \tau_{\mathtt{t}}(i)$. This vertex has neighboring (or offspring) vertices of possibly different types that extend to depth 1, each of which has again neighboring vertices that extend to the next depth, and so on. For a vertex with type $(i, \mathtt{t})$, the number of offspring vertices with type $(j, \mathtt{t}')$ is Poisson distributed with mean $c\eta_{i,j}\gamma_j\tau_{\mathtt{t}'}(j)$, independently of the number of offspring vertices of other types. Since the sum of independent Poisson RVs is again Poisson distributed, we have that the total number of offspring vertices of a vertex with type $(i, \mathtt{t})$ is Poisson distributed with mean

$$\sum_{j=1}^{L} \sum_{\mathtt{t}'=1}^{\mathtt{t}_{\max}} c\gamma_j \eta_{i,j} \tau_{\mathtt{t}'}(j) = c \sum_{j=1}^{L} \gamma_j \eta_{i,j}, \tag{39}$$

independently of $\mathtt{t}$ (cf. (38)). The above multi-type branching process is denoted by $\mathfrak{X}$. We further use $\mathfrak{X}(i, \mathtt{t})$ to denote the process which starts with a root vertex that has the specific type $(i, \mathtt{t})$.

Let $z^{(\ell)}$ be the probability that the root vertex of the first ℓ generations of $\mathfrak{X}$ survives the ℓ iterations of the peeling procedure. This probability is evaluated explicitly in Section V-E in terms of the code parameters $\boldsymbol{\eta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\tau}$. The main result is as follows.

Theorem 3. *Let W_k, $k \in [n]$, be the indicator RV for the event that the k-th component code of $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ declares a decoding failure after ℓ iterations of decoding and define $W = \frac{1}{n}\sum_{k=1}^{n} W_k$. Then, we have*

$$\lim_{n\to\infty} \mathbb{E}[W] = z^{(\ell)}. \tag{40}$$

Furthermore, for any $\varepsilon > 0$, there exist $\delta > 0$, $\beta > 0$, and $n_0 \in \mathbb{N}$ such that for all $n > n_0$ we have

$$\mathbb{P}\left(|W - \mathbb{E}[W]| > \varepsilon\right) \leq e^{-\beta n^{\delta}}. \tag{41}$$

Proof. In order to prove (40), we apply [23, Th. 11.6]. First, recall the following definition from [23, p. 74]. Let $f(v, G)$ be a function defined on a pair (v, G), where G is a graph composed of vertices with different types and v is a distinguished vertex of G, called the root. The function f is an ℓ-neighborhood function if it is invariant under type-preserving rooted-graph isomorphisms and depends only on the neighborhood of the vertex v up to depth ℓ. The RV W_k depends only on the depth-ℓ neighborhood of the k-th vertex in the residual graph of $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$. Furthermore, the peeling outcome for the k-th vertex is invariant under isomorphisms as long as they preserve the vertex type. Hence, the RV W_k can be expressed in terms of an ℓ-neighborhood function $\mathcal{D}_\ell$ as $W_k = \mathcal{D}_\ell(k, G)$. That is, the function $\mathcal{D}_\ell$ evaluates the peeling procedure on the depth-ℓ neighborhood of a vertex and thus determines if the corresponding component code declares a decoding failure after ℓ iterations. To apply [23, Th. 11.6], we need to check that we have $\sup_n \mathbb{E}\left[\mathcal{D}_\ell(k, G)^4\right] < \infty$. This is true since since D_ℓ maps to $\{0, 1\}$. We then have [23, Eq. (11.4)]

$$\lim_{n\to\infty} \mathbb{E}\left[W\right] = \mathbb{E}\left[\mathcal{D}_\ell(\mathfrak{X})\right], \tag{42}$$

where $\mathcal{D}_\ell(\mathfrak{X})$ is defined by evaluating $\mathcal{D}_\ell$ on the branching process $\mathfrak{X}$ up to depth ℓ, taking the initial vertex as the root. Therefore, $\mathbb{E}\left[\mathcal{D}_\ell(\mathfrak{X})\right] = z^{(\ell)}$. This result generalizes the convergence result (4) for HPCs (i.e., $\mathcal{G}(n, c/n)$) shown in Sections IV-A and IV-B. The proof in [23] relies on a stochastic coupling of the branching process $\mathfrak{X}$ and the neighborhood exploration process for $\mathcal{G}^{\mathcal{V}}(n, \boldsymbol{\kappa})$ (which generalizes the exploration process described in Section III-B to handle different vertex types), see [23, Lem. 11.4] for details.

The proof of (41) follows along the same lines as the proof for the homogeneous case in Section IV-D, using again the typical bounded differences inequality. The bound on the maximum vertex degree for $\mathcal{G}(n, c/n)$ in Lemma 1 applies without change also to the inhomogeneous random graph $\mathcal{G}^{\mathcal{V}}(n, \boldsymbol{\kappa})$. The only difference in the proof in Appendix A is that the equality in (93) becomes an inequality. The choice of the high-probability event Γ and the typical Lipschitz constant is then the same as described in Section IV-D. □

E. Density Evolution

In order to compute $z^{(\ell)}$, we proceed in a similar fashion as described in Section IV-C and break down the computation in a recursive fashion. First, note that from the definition of $\mathfrak{X}$ and $\mathfrak{X}(i, \mathtt{t})$, we have

$$z^{(\ell)} = \mathbb{E}\left[\mathcal{D}_\ell(\mathfrak{X})\right] = \sum_{i=1}^{L} \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \gamma_i \tau_{\mathtt{t}}(i) z_{i,\mathtt{t}}^{(\ell)}, \tag{43}$$

where

$$z_{i,\mathtt{t}}^{(\ell)} = \mathbb{E}\left[\mathcal{D}_\ell(\mathfrak{X}(i, \mathtt{t}))\right] \tag{44}$$

is the probability that the root vertex of the first ℓ generations of the branching process $\mathfrak{X}(i, \mathtt{t})$ survives the peeling procedure. We claim that

$$z_{i,\mathtt{t}}^{(\ell)} = \Psi_{\geq \mathtt{t}+1}\left(c \sum_{j=1}^{L} \sum_{\mathtt{t}'=1}^{\mathtt{t}_{\max}} \eta_{i,j} \gamma_j \tau_{\mathtt{t}'}(j) x_{j,\mathtt{t}'}^{(\ell-1)} \right), \tag{45}$$

where $x_{i,\mathtt{t}}^{(\ell)}$ is recursively given by

$$x_{i,\mathtt{t}}^{(\ell)} = \Psi_{\geq \mathtt{t}}\left(c \sum_{j=1}^{L} \sum_{\mathtt{t}'=1}^{\mathtt{t}_{\max}} \eta_{i,j} \gamma_j \tau_{\mathtt{t}'}(j) x_{j,\mathtt{t}'}^{(\ell-1)} \right), \tag{46}$$

with $x_{i,\mathtt{t}}^{(0)} = 1$. The argument is the same as described in Section IV-C. In particular, to determine the survival of the root of $\mathfrak{X}(i, \mathtt{t})$ after ℓ peeling iterations, first determine if each offspring vertex gets removed by applying $\ell - 1$ peeling iterations to the corresponding subtree. Then, make a decision based on the number of surviving offspring vertices. Again, one finds that offspring vertices survive independently of each other, however, the survival probability now depends on the vertex type. In particular, in (45), the quantity $x_{i,\mathtt{t}}^{(\ell-1)}$ is the probability that a type-$(i, \mathtt{t})$ offspring of the root vertex survives the $\ell - 1$ peeling iterations applied to its subtree. The argument of the function $\Psi_{\geq \mathtt{t}+1}$ in (45) is the mean number of surviving offspring vertices, which, again, is easily found to be Poisson distributed. Essentially the same arguments can be applied to find (46) by taking into account the connecting edge of each offspring vertex to the previous level of the tree.

Using the substitution $x_i^{(\ell)} = \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \tau_{\mathtt{t}}(i) x_{i,\mathtt{t}}^{(\ell)}$, it is often more convenient to express $z^{(\ell)}$ in terms of

$$z^{(\ell)} = \sum_{i=1}^{L} \gamma_i \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \tau_{\mathtt{t}}(i) \Psi_{\geq \mathtt{t}+1}\left(c \sum_{j=1}^{L} \eta_{i,j} \gamma_j x_j^{(\ell-1)} \right), \tag{47}$$

where

$$x_i^{(\ell)} = \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \tau_{\mathtt{t}}(i) \Psi_{\geq \mathtt{t}}\left(c \sum_{j=1}^{L} \eta_{i,j} \gamma_j x_j^{(\ell-1)} \right) \tag{48}$$

with $x_i^{(0)} = 1$ for all $i \in [L]$.

VI. Discussion

Before considering a direct application of the obtained DE equations in the next section, we briefly discuss some relevant topics regarding their general application.

A. Thresholds and Code Comparisons

The decoding threshold for $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ can be defined in terms of the effective channel quality as

$$c^* \triangleq \sup\{c > 0 \,|\, \lim_{\ell \to \infty} z^{(\ell)} = 0\}. \tag{49}$$

Recall that in the code construction in Section V-A, $\boldsymbol{\gamma}$ is assumed to be a distribution, i.e., we have $\sum_{i=1}^{L} \gamma_i = 1$. This assumption turns out to be convenient in the formulation and proof of Theorem 3 since it ensures that the number of CNs in the Tanner graph is always given by n. However, when comparing the performance of different GPCs (for example in terms of thresholds computed via (49)), it is more appropriate to lift this assumption and replace γ_i by a rescaled version $a\gamma_i$ for all i and some constant a. This simply corresponds to scaling the total number of CNs to an.

A reasonable scaling to compare different codes is to choose a such that the effective channel quality c can be interpreted asymptotically as the average number of initial erasures in each component code, similar to HPCs in Section II-C. Since each component code at position i initially contains $an_{\mathcal{C},i}c/n$ erasures, by averaging over all positions we obtain

$$\lim_{n \to \infty} a \frac{c}{n} \sum_{i=1}^{L} \gamma_i \left(\sum_{j \neq i} \gamma_j n \eta_{i,j} + \eta_{i,i}(\gamma_i n - 1) \right) \tag{50}$$

$$= ac \sum_{i=1}^{L} \gamma_i \sum_{j=1}^{L} \gamma_j \eta_{i,j} = ac \boldsymbol{\gamma}^{\mathsf{T}} \boldsymbol{\eta} \boldsymbol{\gamma}. \tag{51}$$

Setting (51) equal to c leads to

$$a = \frac{1}{\boldsymbol{\gamma}^{\mathsf{T}} \boldsymbol{\eta} \boldsymbol{\gamma}}. \tag{52}$$

Example 14. For staircase codes (see Example 10), we obtain $a = (2L-2)/L^2$. For large L, $a \approx 2/L$ so that $a\gamma_i \approx 1/2$. △

B. Upper Bound on the Decoding Threshold

An upper bound on the decoding threshold for $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ can be given as follows, see [31, Sec. VI-A]. Assume for a moment that all component codes can correct up to $\mathtt{t}$ erasures. The best one can hope for in this case is that each component code corrects exactly $\mathtt{t}$ erasures. That is, in total at most $a\mathtt{t}n$ erasures can be corrected, where a is assumed to be defined as in (52). Normalizing by the code length m (see (32)) gives a maximum erasure probability of $p \leq a\mathtt{t}n/m$ or, in terms of the effective channel quality $c \leq a\mathtt{t}n^2/m$. Using (32)

as $n \to \infty$, we obtain $c \leq 2\mathsf{t}$ as a necessary condition for successful decoding. This reasoning extends naturally also to the case where we allow for a mixture of erasure-correcting capabilities. In this case, one finds that $c \leq 2\bar{\mathsf{t}}$, where

$$\bar{\mathsf{t}} = \sum_{i=1}^{L} \gamma_i \sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}}(i)\mathsf{t} \tag{53}$$

is the mean erasure-correcting capability. This bound is used for example as a reference in the code optimization discussed in Section VII.

Remark 16. A similar discussion can be found in [33], where the authors refer to the resulting threshold bound (see [33, Def. 1]) as the "weight-pulling" threshold.

C. Modified Decoding Schedules

We now discuss decoding algorithms that differ from the one described in Section II-C and Section V-C in terms of scheduling. For example, for conventional PCs, one typically iterates between the component decoders for the row and column codes. Another example is the decoding of convolutional-like GPCs, such as the ones described in Examples 10 and 11. For these codes, L is typically assumed to be very large and it becomes customary to employ a sliding-window decoder. Such a decoder does not require knowledge of the entire received code array in order to start decoding. The decoder instead only operates on a subset of the array within a so-called window configuration. After a predetermined number of iterations, this subset changes and the window "slides" to the next position.

More generally, assume that we wish to apply a different decoding schedule to $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$. To that end, let $\mathcal{A}^{(l)} \subseteq [L]$ for $l \in [\ell]$ be a subset of the L CN positions. We interpret $\mathcal{A}^{(l)}$ as *active* positions and the complementary set $[L] \setminus \mathcal{A}^{(l)}$ as *inactive* positions in iteration l. The decoding is modified as follows. In iteration l, one only executes the BDD corresponding to CNs at active positions, i.e., positions that are contained in the set $\mathcal{A}^{(l)}$. CNs at inactive positions are assumed to be frozen, in the sense that they do not contribute to the decoding process. In the peeling procedure, vertices at inactive positions are simply ignored during iteration l.

In order to check if Theorem 3 remains valid for a modified decoding schedule, we adopt the convention that frozen CNs continue to declare a decoding failure if they declared a failure in the iteration in which they were last active. Moreover, we assume that each CN position belongs to the set of active positions at least once during the decoding, i.e., we assume that $\bigcup_{l=1}^{\ell} \mathcal{A}^{(l)} = [L]$ (otherwise W_k in Theorem 3 is not defined for CNs that were never activated). Using these assumptions, it can be shown that Theorem 3 remains valid. The only difference is that the corresponding DE equations now depend on the schedule through

$$z_i^{(\ell)} = \begin{cases} \text{RHS of (47)} & \text{if } i \in \mathcal{A}^{(\ell)} \\ z_i^{(\ell-1)} & \text{otherwise} \end{cases}, \tag{54}$$

and

$$x_i^{(\ell)} = \begin{cases} \text{RHS of (48)} & \text{if } i \in \mathcal{A}^{(\ell)} \\ x_i^{(\ell-1)} & \text{otherwise} \end{cases}. \tag{55}$$

To see this first observe that in the proof of Theorem 3, the decoding schedule can be handled by simply assuming an appropriately modified neighborhood function $\tilde{\mathcal{D}}_\ell$. In particular, one may think about embedding the decoding schedule $(\mathcal{A}^{(l)})_{l\in[\ell]}$ into the function $\tilde{\mathcal{D}}_\ell$. Observe that the scheduling does not change the fact that the decoding outcome is isomorphism invariant, as long as the type of all vertices is preserved. Thus, it remains to show that applying the modified decoding function $\tilde{\mathcal{D}}_\ell$ to the branching process $\mathfrak{X}$ results in (54) and (55). Assuming that the root vertex is active in the final iteration ℓ, we can proceed as before. If, on the other hand, the root vertex is not active in the final iteration ℓ, we know that the survival probability is the same as it was in the previous iteration. This gives (54) and applying the same reasoning for offspring vertices gives (55).

D. Performance on the Binary Symmetric Channel

When assuming transmission over the binary symmetric channel (BSC) as opposed to the BEC, the crucial difference is that there is a possibility that the component decoders may miscorrect, in the sense that they introduce additional errors into the iterative decoding process. This makes a rigorous analysis challenging.

One possible approach is to change the iterative decoder. In particular, consider again the message-passing interpretation of the iterative decoding in Remark 3. In [31], the authors propose to modify the decoder in order to make the corresponding message-passing update rules extrinsic. In this case, miscorrections can be rigorously incorporated into the asymptotic decoding analysis for GPC ensembles. The reason why this approach works from a DE perspective is that for code ensembles, the entire computation graph (for a fixed depth) of a CN in the Tanner graph becomes tree-like. In fact, this makes it possible to analyze a variety of extrinsic message-passing decoders for a variety of different channels [17], including the above modified iterative decoder for the BSC.

Unfortunately, this approach appears to be limited to code ensembles. Recall that for the deterministic GPC construction $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$, it is only the neighborhood in the residual graph that becomes tree-like (not the entire computation graph). Therefore, the independence assumption between messages is not necessarily satisfied, neither for intrinsic nor extrinsic message-passing algorithms. In general, it is not obvious how to rigorously incorporate miscorrections into an asymptotic analysis for a deterministic GPC construction. Applying our results to the BSC thus requires a similar assumption as in [9], [11], [19], i.e., either one assumes that miscorrections are negligible or that a genie prevents them.

E. Spatially-Coupled Product Codes

Of particular interest are GPCs where the matrix $\boldsymbol{\eta}$ has a band-diagonal "convolutional-like" structure. The associated GPC can then be classified as a spatially-coupled PC. For example, the GPCs discussed in Examples 10 and 11, i.e., staircase and braided codes, are particular instances of spatially-coupled PCs. Spatially-coupled codes have attracted

a lot of attention in the literature due to their outstanding performance under iterative decoding [50], [51].

In [52], we study the asymptotic performance of deterministic spatially-coupled PCs based on the theory derived in this paper. In particular, we provide a detailed comparison to spatially-coupled PCs that are based on the code ensembles proposed in [31]–[33]. One of the main outcomes of this work is that there exists a family of deterministic spatially-coupled PCs (see [52, Def. 2]) that asymptotically follows the same DE recursion as the ensembles defined in [31]–[33]. This implies that certain ensemble properties proved in these papers (in particular lower bounds on the decoding thresholds via potential function methods) also apply to the deterministic code family as well. An important step to show this result is to transform the ensemble DE recursions obtained in [31]–[33] into a form that makes them comparable to the DE recursion for deterministic GPCs obtained in this paper. We also show that there exists a related, but structurally simpler, deterministic code family (see [52, Def. 5]) that attains essentially the same asymptotic performance. The simpler code family follows a slightly different DE recursion, which can also be analyzed using potential function methods (see the appendix of [52] for details).

It is also interesting to compare the asymptotic DE predictions to performance of practical finite-length spatially-coupled PCs. This was done in [53] for staircase, braided, and so-called half-braided codes assuming a window decoder with realistic parameters.

F. Parallel Binary Erasure Channels

It is possible to generalize the analysis presented in this paper to the case where transmission takes place over M parallel BECs with different erasure-correcting capabilities $p_1, \dots, p_M$ [54]. In that scenario, one considers the scaling $p_1 = c_1/n, \dots, p_M = c_M/n$, where the positive constants $c_1, \dots, c_M$ act as the effective channel qualities for the parallel BECs. As an application, the authors showed in [54] that the resulting asymptotic performance analysis can be used to predict and optimize the performance of $\mathcal{C}_n(\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\tau})$ when combined with a higher-order signal constellation in a coded modulation setup assuming a hard-decision symbol detector.

VII. Irregular Half-Product Codes

In this section, we consider an application of the derived DE equations for deterministic GPCs. In particular, we discuss the optimization of component code mixtures for HPCs. Recall that for (regular) HPCs, we have $\boldsymbol{\eta} = 1$, $\boldsymbol{\gamma} = 1$, and all component codes associated with the CNs have the same erasure-correcting capability t. Similar to irregular PCs [20], [21], an *irregular* HPC is obtained by assigning component codes with different erasure-correcting capabilities to the CNs. The primary goal of this section is to show how the asymptotic DE analysis can be used in practice to achieve performance improvements for finite-length codes. The general approach is to use decoding thresholds as an optimization criterion. This is, of course, completely analogous to optimizing degree distributions of irregular LDPC codes based on DE. Thus, it comes with similar caveats (e.g., no optimality guarantees for finite code lengths), but also with similar strengths (e.g., an efficient and fast optimization procedure).

A. Preliminaries

The assignment of erasure-correcting capabilities to the CNs is done according to the distribution $\boldsymbol{\tau} = (\tau_1, \dots, \tau_{\mathsf{t}_{\max}})^\intercal$. (For notational convenience, we suppress the dependence of the distribution and other quantities on the position index in the Tanner graph.) The mean erasure-correcting capability (53) in this case is given by

$$\bar{\mathsf{t}} = \sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} \mathsf{t}. \tag{56}$$

The DE equation (48) simplifies to

$$x^{(\ell)} = \sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} \Psi_{\geq \mathsf{t}}(c x^{(\ell-1)}), \tag{57}$$

with $x^{(0)} = 1$. The decoding threshold (49) can alternatively be written as

$$c^* = \sup\{c > 0 \,|\, \lim_{\ell \to \infty} x^{(\ell)} = 0\}, \tag{58}$$

since $z^{(\ell)} \to 0$ if and only if $x^{(\ell)} \to 0$ as $\ell \to \infty$. From (57) and the fact that $\Psi_{\geq \mathsf{t}}(x)$ for any $\mathsf{t} \in \mathbb{N}$ and $x \geq 0$ is strictly increasing, we have that the condition

$$\sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} \Psi_{\geq \mathsf{t}}(cx) < x, \qquad \text{for } x \in (0, 1], \tag{59}$$

implies successful decoding after a sufficiently large number of iterations, i.e., we have that $c^* \geq c$.

We wish to design $\boldsymbol{\tau}$ such that c^* is as large as possible. Obviously, choosing component codes with larger erasure-correcting capability gives better performance, i.e., larger thresholds. Thus, the design is done under the constraint that the mean erasure-correcting capability $\bar{\mathsf{t}}$ remains fixed. This is the natural analogue to the rate-constraint when designing degree distributions for irregular LDPC codes.

B. Lower Bounds on the Threshold

Before discussing the practical optimization of the distribution $\boldsymbol{\tau}$ based on a linear program in the next subsection, we show that one can construct irregular HPCs that have thresholds

$$2\bar{\mathsf{t}} - 1 \leq c^* \leq 2\bar{\mathsf{t}}, \tag{60}$$

where we recall that the upper bound in (60) holds for any GPC according to the discussion in Section VI-B. The lower bound in (60) is achieved by a uniform distribution. In particular, from $\sum_{i=1}^{\infty} \mathbb{P}(X \geq i) = \mathbb{E}[X]$, we have

$$\sum_{\mathsf{t}=1}^{\infty} \Psi_{\geq \mathsf{t}}(cx) = cx, \tag{61}$$

where we recall that $\Psi_{\geq \mathsf{t}}(cx) = \mathbb{P}\left(\mathsf{Po}(cx) \geq \mathsf{t}\right)$. If we then choose a uniform distribution according to $\tau_{\mathsf{t}} = 1/N$ for $\mathsf{t} \in [N]$ (i.e., $\mathsf{t}_{\max} = N$), we have

$$\sum_{\mathsf{t}=1}^{N} \tau_{\mathsf{t}} \Psi_{\geq \mathsf{t}}(Nx) < \sum_{\mathsf{t}=1}^{\infty} \frac{1}{N} \Psi_{\geq \mathsf{t}}(Nx) = x \quad \text{for } x > 0, \tag{62}$$

where the (strict) inequality follows from the fact that $\Psi_{\geq \mathsf{t}}(x) > 0$ for any $\mathsf{t} \in \mathbb{N}$ and $x > 0$. We see from (62) that the threshold for the uniform distribution satisfies $c^* \geq N$ (cf. (59)). Moreover, the average erasure-correcting capability is given by

$$\bar{\mathsf{t}} = \sum_{\mathsf{t}=1}^{N} \tau_{\mathsf{t}} \mathsf{t} = \frac{1}{N} \frac{N(N+1)}{2} = \frac{N+1}{2}. \tag{63}$$

Therefore, we have

$$2\bar{\mathsf{t}} - c^* \leq 2\frac{N+1}{2} - N = 1, \tag{64}$$

or $c^* \geq 2\bar{\mathsf{t}} - 1$. This simple lower bound shows that one can design irregular HPCs that are within a constant gap of the upper $2\bar{\mathsf{t}}$-bound. This is in contrast to regular HPCs where $\bar{\mathsf{t}} = \mathsf{t}$. In this case, the difference between the threshold c^* and $2\mathsf{t}$ becomes unbounded for large t, since $c^* = \mathsf{t} + \sqrt{\mathsf{t} \log \mathsf{t}} + \mathcal{O}(\log(\mathsf{t}))$ [34].

Remark 17. Essentially the same argument also allows us to give a lower bound on the threshold for irregular HPCs when the minimum erasure-correcting capability is constrained to some value $\mathsf{t}_{\min} > 1$. In that case, a uniform distribution over $\{\mathsf{t}_{\min}, \mathsf{t}_{\min}+1, \ldots, \mathsf{t}_{\min}+N-1\}$ still gives a threshold that satisfies $c^* \geq N$. However, we have $\bar{\mathsf{t}} = (N + 2\mathsf{t}_{\min} - 1)/2$. Hence, one obtains the lower bound $c^* \geq 2\bar{\mathsf{t}} - 2\mathsf{t}_{\min} + 1$.

C. Optimization via Linear Programming

The optimal distribution maximizes the threshold c^* subject to a fixed mean erasure-correcting capability $\bar{\mathsf{t}}$. Alternatively, one may fix a certain channel quality parameter c and minimize $\bar{\mathsf{t}}$ as follows.

$$\underset{\tau_1, \ldots, \tau_{\mathsf{t}_{\max}}}{\text{minimize}} \quad \bar{\mathsf{t}} = \sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} \mathsf{t} \tag{65}$$

$$\text{subject to} \quad \sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} = 1, \quad \tau_1, \ldots, \tau_{\mathsf{t}_{\max}} \geq 0 \tag{66}$$

$$\sum_{\mathsf{t}=1}^{\mathsf{t}_{\max}} \tau_{\mathsf{t}} \Psi_{\geq \mathsf{t}}(cx) < x, \quad x \in (0, 1]. \tag{67}$$

The objective function and all constraints in (65)–(67) are linear in $\tau_1, \ldots, \tau_{\mathsf{t}_{\max}}$. Thus, after discretizing the constraint (67) according to $x = i\Delta$ for $i \in [M]$ and $\Delta = 1/M$, one obtains a linear program, which can be efficiently solved by standard numerical optimization solvers. In Fig. 8, we show the thresholds of the optimized irregular HPCs by the red line, where we used $M = 1000$ and $\mathsf{t}_{\max} = 50$, as a function of $\bar{\mathsf{t}}$. We also show the thresholds for regular HPCs (where $\bar{\mathsf{t}} = \mathsf{t} = 2, 3, \ldots$) and the $2\bar{\mathsf{t}}$-bound by the blue and black lines, respectively. It can be seen that the thresholds for regular HPCs diverge from the bound for large $\bar{\mathsf{t}}$, as expected. Using

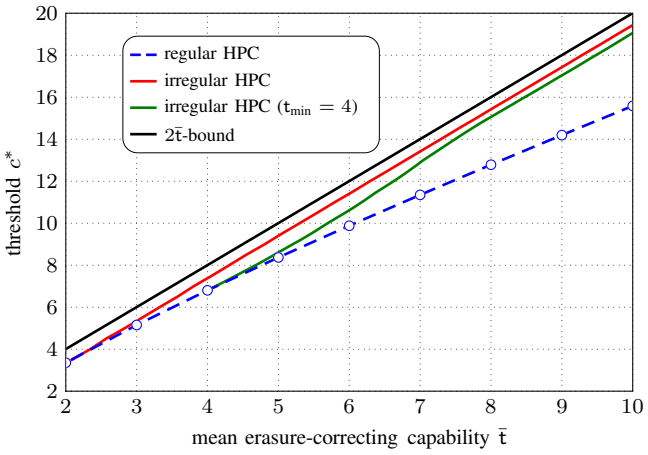


Fig. 8. Decoding thresholds for optimized irregular and regular HPCs. Thresholds for irregular HPCs are obtained via a discretized linear program with $M = 1000$ and $\mathsf{t}_{\max} = 50$.

irregular HPCs, the thresholds can be significantly improved for large $\bar{\mathsf{t}}$. However, there appears to be an almost constant gap between the upper bound and the threshold curve. This gap is investigated in more detail in the next subsection.

For practical applications, it is often desirable to limit the fraction of component codes with "small" erasure-correcting capabilities in order to avoid harmful error floors [11]. It is straightforward to incorporate a minimum erasure-correcting capability $\mathsf{t}_{\min}$ into the above linear program. For example, the green line in Fig. 8 shows the thresholds of the optimized irregular HPCs when the minimum erasure-correcting capability is constrained to $\mathsf{t}_{\min} = 4$. This additional constraint entails a threshold penalty which, however, decreases for larger values of $\bar{\mathsf{t}}$.

D. Initial Component Code Loss

We now focus in more detail on the upper $2\bar{\mathsf{t}}$-bound for the thresholds of irregular HPCs. In particular, we show that it is possible to give a slightly improved upper bound based on the notion of an *initial component code loss*. Based on this, it can be shown that the upper bound in (60) is in fact strict, i.e., for any distribution $\boldsymbol{\tau}$ with mean erasure-correcting capability $\bar{\mathsf{t}}$ and threshold c^*, the gap $2\bar{\mathsf{t}} - c^*$ is always bounded away from zero. This gives an intuitive explanation for the gap between the threshold curve and the $2\bar{\mathsf{t}}$-bound observed in Fig. 8. A similar bound for irregular LDPC code ensembles over the BEC is given in [55].

Recall that the upper bound has been derived in Section VI-B under the (somewhat optimistic) assumption that each t-erasure correcting component code corrects exactly t erasures. In other words, each component code is assumed to contribute its maximum erasure-correcting potential to the overall decoding. A refined version of this argument takes into account the fact that a certain amount of erasure-correcting potential is lost almost surely before the iterative decoding process even begins. In particular, let the RVs $N_{i,\mathsf{t}}$, for $i = 0, 1 \ldots, \mathsf{t} - 1$, be the number of CNs corresponding to

$\mathtt{t}$-erasure-correcting component codes that are initially connected to $i < \mathtt{t}$ erased VNs. In the first decoding iteration, each of these CNs corrects only i erasures instead of $\mathtt{t}$, i.e., the maximum number of erasures E that we can hope to correct is upper bounded by

$$E \leq \bar{\mathtt{t}} n - \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \sum_{i=0}^{\mathtt{t}-1} N_{i,\mathtt{t}}(\mathtt{t}-i). \tag{68}$$

Since E/n and $N_{i,\mathtt{t}}/n$ converge almost surely to the deterministic values $c/2$ and $\tau_{\mathtt{t}} \Psi_{=i}(c)$, respectively, we obtain

$$c \leq 2\bar{\mathtt{t}} - 2\mathcal{L}_{\boldsymbol{\tau}}(c) \tag{69}$$

as a necessary condition for successful decoding, where we implicitly defined the initial component code loss for the distribution $\boldsymbol{\tau}$ as

$$\mathcal{L}_{\boldsymbol{\tau}}(c) \triangleq \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \tau_{\mathtt{t}} \mathcal{L}(\mathtt{t}, c) \tag{70}$$

with

$$\mathcal{L}(\mathtt{t}, c) \triangleq \sum_{i=0}^{\mathtt{t}-1} \Psi_{=i}(c)(\mathtt{t}-i) \tag{71}$$

for $c > 0$ and $\mathtt{t} \in \mathbb{N}$.

Remark 18. The affine extension of $\mathcal{L}(\mathtt{t}, c)$ for a fixed $c \geq 0$ is convex in $\mathtt{t} \in [1; \infty)$ in the sense that for any $c \geq 0$ and $\mathtt{t} = 2, 3, \ldots$, we have

$$\mathcal{L}(\mathtt{t}-1, c) + \mathcal{L}(\mathtt{t}+1, c) = 2\mathcal{L}(\mathtt{t}, c) + \Psi_{=t}(c) \tag{72}$$

$$\geq 2\mathcal{L}(\mathtt{t}, c). \tag{73}$$

This implies that for any distribution $\boldsymbol{\tau}$ with average erasure-correcting capability $\bar{\mathtt{t}}$, the associated initial component code loss satisfies

$$\mathcal{L}_{\boldsymbol{\tau}}(c) \geq \mathcal{L}(\lfloor \bar{\mathtt{t}} \rfloor, c), \tag{74}$$

i.e., the initial loss is minimized for regular HPCs.

The bound (69) has a natural interpretation in terms of areas related to the curves involved in the condition (59), similar to the area theorem for irregular LDPC code ensembles. Indeed, an alternative way to show that successful decoding implies (69) is by integrating the condition (59). Using integration by parts, one obtains the indefinite integral [56]

$$\int \Psi_{\geq \mathtt{t}}(x)\, \mathrm{d}x = x\Psi_{\geq \mathtt{t}}(x) + \mathtt{t}\Psi_{\leq \mathtt{t}}(x). \tag{75}$$

Thus, we have

$$c \int_0^1 \Psi_{\geq \mathtt{t}}(cx)\, \mathrm{d}x = c\Psi_{\geq \mathtt{t}}(c) + \mathtt{t}\Psi_{\leq \mathtt{t}}(c) - \mathtt{t} \tag{76}$$

$$= c(1 - \Psi_{<\mathtt{t}}(c)) + \mathtt{t}\Psi_{\leq \mathtt{t}}(c) - \mathtt{t} \tag{77}$$

$$= c - \mathtt{t} + \mathcal{L}(\mathtt{t}, c), \tag{78}$$

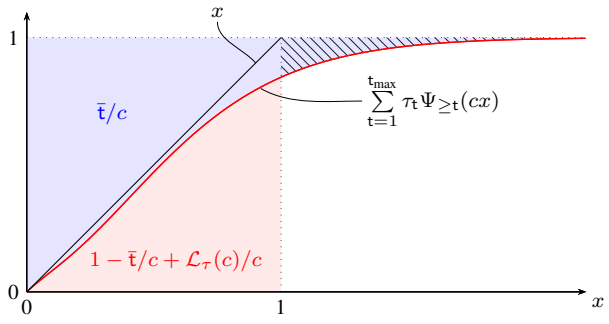


Fig. 9. Graphical interpretation of the upper threshold bounds.

where the last equality follows from

$$\mathtt{t}\Psi_{\leq \mathtt{t}}(c) - c\Psi_{<\mathtt{t}}(c) = \mathtt{t}\Psi_{\leq \mathtt{t}}(c) - c\sum_{k=0}^{\mathtt{t}-1} \frac{c^k}{k!} e^{-c} \tag{79}$$

$$= \mathtt{t}\Psi_{\leq \mathtt{t}}(c) - c\sum_{k=1}^{\mathtt{t}} \frac{c^{k-1}}{(k-1)!} e^{-c} \tag{80}$$

$$= \mathtt{t}\Psi_{\leq \mathtt{t}}(c) - \sum_{k=0}^{\mathtt{t}} \frac{c^k}{k!} e^{-c} k \tag{81}$$

$$= \sum_{k=0}^{\mathtt{t}} \Psi_{=k}(c)(\mathtt{t}-k) \tag{82}$$

$$= \mathcal{L}(\mathtt{t}, c). \tag{83}$$

Hence, integrating both sides of (59) from zero to one and using (78), one obtains

$$\frac{1}{c} \sum_{\mathtt{t}=1}^{\mathtt{t}_{\max}} \tau_{\mathtt{t}} \left(c - \mathtt{t} + \mathcal{L}(\mathtt{t}, c)\right) < \frac{1}{2}, \tag{84}$$

or, equivalently, (69).

A visualization is shown in Fig. 9, where the red and black lines correspond to the left-hand side (LHS) and RHS of (59), respectively. The area below the red curve up to $x = 1$ (shown in red) corresponds to the LHS of (84). Similarly, it can be shown using (75) that the area between the red line and $x = 1$ (shown in blue) corresponds to the scaled erasure-correcting capability $\bar{\mathtt{t}}/c$. Note that the $2\bar{\mathtt{t}}$-bound on the threshold simply corresponds to the fact that the blue area cannot be smaller than $1/2$, since otherwise the red and black lines would have to cross. From the previous discussion, we have seen that the gap to the upper $2\bar{\mathtt{t}}$-bound is partially due to the initial component code loss. In particular, by combining the blue and red areas, it can be seen that the hatched area in Fig. 9 corresponds precisely to the (scaled) loss $\mathcal{L}_{\boldsymbol{\tau}}(c)/c$.

Consider now again the outcome of the linear program for the optimized irregular HPCs in Fig. 8. In Fig. 10, the (vertical) gap $2\bar{\mathtt{t}} - c^*$ between the black and red lines in Fig. 8 is shown for a larger range of $\bar{\mathtt{t}}$. It can be seen that the gap is decreasing with $\bar{\mathtt{t}}$, albeit rather slowly. We also plot the initial component code loss for the optimized distributions at the threshold value by the blue line. From this, we see that the initial component code loss accounts for approximately half of the threshold gap for the optimized irregular distributions.

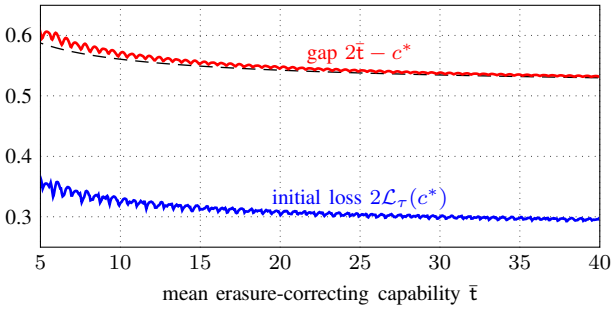


Fig. 10. Gap between the threshold c^* and the $2\bar{\mathsf{t}}$-bound for the optimized irregular HPCs.

Remark 19. In fact, we conjecture that the following is true. Assume $c \in \mathbb{N}$. Then, for any distribution $\boldsymbol{\tau}$ with threshold $c^* \geq c$ and mean erasure-correcting capability $\bar{\mathsf{t}}$, we have

$$\bar{\mathsf{t}} \geq \frac{c}{2} + \frac{1}{c} \sum_{\mathsf{t}=1}^{c} \mathcal{L}(\mathsf{t}, c). \tag{85}$$

This bound is shown in Fig. 10 by the dashed line, although we failed to prove it. Proving (85) would be interesting, since one can show that $\lim_{c\to\infty} \frac{1}{c} \sum_{\mathsf{t}=1}^{c} \mathcal{L}(\mathsf{t}, c) = 1/4$ and hence $2\bar{\mathsf{t}} - c^* \geq 1/2$, which seems to be the constant to which the optimization outcome is converging for $\bar{\mathsf{t}} \to \infty$.

E. Simulation Results

In order to illustrate how the thresholds can be used to design practical irregular HPCs, we consider (shortened) binary BCH codes as component codes. Given the Galois-field extension degree ν, a shortening parameter s, and the erasure-correcting capability t, we let the component code be an $(n, k_{\mathcal{C}}, d_{\min})$ BCH code, where $n = 2^\nu - 1 - s$, $d_{\min} = \mathsf{t}+1$, and

$$k_{\mathcal{C}} = \begin{cases} n - \nu\mathsf{t}/2, & \mathsf{t} \text{ even} \\ n - \nu(\mathsf{t}-1)/2 - 1, & \mathsf{t} \text{ odd} \end{cases}. \tag{86}$$

In the following, we consider two irregular HPCs, where $\bar{\mathsf{t}} \approx 7$. As a comparison, we use a regular HPC with $\tau_7 = 1$ for which $c^* \approx 11.34$. The optimal distribution (rounded to three decimal places) according to the linear program (65)–(67) is given by

$$\begin{aligned} &\tau_1 = 0.070, \quad \tau_2 = 0.103, \quad \tau_4 = 0.115, \\ &\tau_5 = 0.179, \quad \tau_{10} = 0.496, \quad \tau_{11} = 0.037, \end{aligned} \tag{87}$$

which yields $c^* \approx 13.42$. We also consider the case where the minimum erasure-correcting capability is constrained to be $\mathsf{t}_{\min} = 4$. For this case, one obtains

$$\tau_4 = 0.495, \quad \tau_9 = 0.029, \quad \tau_{10} = 0.476, \tag{88}$$

and the threshold is reduced to $c^* \approx 12.88$.

For the simulations, we consider two different component code lengths, $n = 1000$ (i.e., $\nu = 10$ and $s = 23$) and $n = 3000$ (i.e., $\nu = 12$ and $s = 1095$), leading to an overall length of the HPCs of $m \approx 500,000$ and $m \approx 4,500,000$,

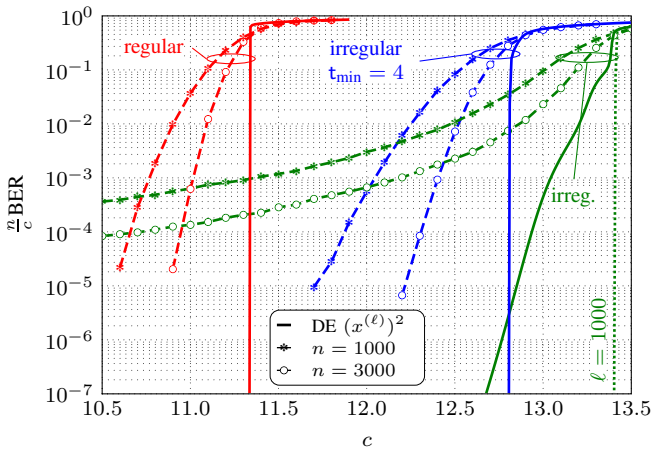


Fig. 11. Simulation results (dashed lines) for regular and optimized irregular HPCs for two different values of n and $\ell = 100$. DE results (solid lines) are shown for $\ell = 100$.

respectively. If we denote the dimension of the k-th component code by $k_{\mathcal{C}_k}$, the code rate is lower bounded by [7, Sec. 5.2.1]

$$R \geq 1 - \frac{\sum_{k=1}^{n} (n - k_{\mathcal{C}_k})}{m}. \tag{89}$$

For the regular case and the distributions (87) and (88), the lower bound evaluates to approximately 0.93 and 0.97 for $n = 1000$ and $n = 3000$, respectively. (In order to obtain shorter (longer) codes for the same rate, one needs to reduce (increase) $\bar{\mathsf{t}}$.) Although the chosen values for n are merely for illustration purposes, we remark that the delay caused by such seemingly long block-lengths is typically not a problem for high-speed applications. For example, the delay for the GPCs designed for fiber-optical communication systems in [8], [13] is in the order of $2,000,000$ bits.

Simulation results are shown in Fig. 11 by the dashed lines. In all cases, the maximum number of decoding iterations is restricted to $\ell = 100$. Results for regular HPCs are shown in red, while results for the irregular HPCs defined by the optimized distributions (87) and (88) are shown in green and blue, respectively. For lower error rates, the irregular HPCs defined by (87) are clearly outperformed by regular HPCs and HPCs defined by the distribution (88). This is due to the relatively large fraction of component codes that only correct 1 and 2 erasures, which leads to a large error floor.

It is interesting to inspect the DE predictions for $\ell = 100$, which are shown by the solid lines in Fig. 11. The predicted performance for the regular and irregular distribution (88) drops sharply, while the predicted performance for the distribution (87) shows a markedly different behavior due to the finite iteration number. It is therefore important to stress that an optimization via the condition (59) implicitly assumes an unrestricted number of decoding iterations. (As a reference, the DE prediction for the distribution (87) with $\ell = 1000$ is shown by the green dotted line.) Thus, if we had done an optimization based on DE assuming $\ell = 100$ and targeting an error rate of around 10^{-7} in Fig. 11, we would have rejected the distribution (87) in favor of the distribution (88) right away. A method to incoorporate the number of decoding

iterations into the threshold optimization of irregular LDPC code ensembles is discussed for example in [57].

Lastly, the HPCs defined by (88) have a comparable finite-length scaling behavior below the threshold and no noticeable error-floor for the simulated error rates. As a consequence, the performance gains for this distribution over the regular HPCs predicted by DE are well preserved also for finite lengths.

VIII. Conclusions and Future Work

In this paper, we studied the performance of deterministically constructed GPCs under iterative decoding. Using the framework of sparse inhomogeneous random graphs, we showed how to derive the DE equations that govern the asymptotic behavior. In principle, DE can be used for a variety of different applications, e.g., parameter tuning, optimization of decoding schedules, or the design of new GPCs. Here, we used the derived DE equations to optimize irregular HPCs that employ a mixture of component codes with different erasure-correcting capabilities. Using an approach based on linear programming, we obtained irregular HPCs that outperform regular HPCs.

For future work, it would be interesting to analyze deterministic code constructions that incorporate VNs with larger degrees. Larger VN degrees are easily incorporated into an ensemble approach, see, e.g., [33]. An example of a corresponding deterministic code construction is the case where code arrays are generalized from two to three (or higher) dimensional objects, e.g., a cube-shaped code array. In that case, the residual graph could be modeled as a random hypergraph. Cores in random hypergraphs have for example been studied in [58].

Another interesting topic for future work is the investigation of the finite-length scaling behavior of deterministic GPCs, similar to the work for LDPC codes in [59]. In fact, the scaling behavior of the k-core in random graphs is characterized in [60] and it would be interesting to translate these results into the coding setting. To the best of our knowledge, finite-length scaling for the inhomogeneous random graph model has not yet been considered. However, such an analysis would be of great practical value since it may allow for accurate performance estimations of GPCs under iterative BDD for finite values of n.

Appendix A
Proof of Lemma 1

First, we upper-bound the probability that the degree D_k of the k-th vertex exceeds d_n using the Chernoff bound. Let $\theta \sim \mathsf{B}(c/n)$, then, for any $\lambda > 0$, we have

$$\begin{aligned}
\mathbb{P}\left(D_k \geq d_n\right) &= \mathbb{P}\left(e^{\lambda D_k} \geq e^{\lambda d_n}\right) && (90)\\
&\leq e^{-\lambda d_n}\mathbb{E}\left[e^{\lambda D_k}\right] && (91)\\
&= e^{-\lambda d_n}\mathbb{E}\left[e^{\lambda(\theta_{k,1}+\cdots+\theta_{k,n})}\right] && (92)\\
&= e^{-\lambda d_n}\left(\mathbb{E}\left[e^{\lambda\theta}\right]\right)^{n-1} && (93)\\
&= e^{-\lambda d_n}(1-p+pe^{\lambda})^{n-1} && (94)\\
&\leq e^{-\lambda d_n}\left(1+\frac{c}{n}\left(e^{\lambda}-1\right)\right)^{n} && (95)\\
&\leq e^{-\lambda d_n}e^{c(e^{\lambda}-1)} && (96)\\
&\leq e^{-c-d_n\ln\frac{d_n}{ce}} && (97)
\end{aligned}$$

where (91) follows from applying Markov's inequality, (93) holds because all $\theta_{k,j} \sim \theta$ are independent except $\theta_{k,k} = 0$, (96) stems from $(1+x/n)^n \leq e^x$ for $x \geq 0$, and (97) follows from minimizing over λ. Thus, for $d_n = \Omega(\log(n))$ and any $\beta > 0$, there is an n_0 such that $\mathbb{P}\left(D_k \geq d_n\right) \leq e^{-\beta d_n}$. Hence, if one chooses β large enough, then the union bound implies

$$\begin{aligned}
\mathbb{P}\left(D_{\max} \geq d_n\right) &\leq n\mathbb{P}\left(D_k \geq d_n\right) && (98)\\
&\leq ne^{-\beta d_n} && (99)\\
&= e^{\log(n)-\beta d_n} && (100)\\
&= e^{-\beta(d_n-\log(n)/\beta)} && (101)\\
&\leq e^{-\beta d_n/2} && (102)
\end{aligned}$$

for all $n \geq n_0$.

Appendix B
Bound on the Second Moment of T_ℓ

To obtain a bound on $\mathbb{E}[T_\ell^2]$, we first show how to compute the corresponding quantity $\mathbb{E}[\bar{T}_\ell^2]$ for the branching process. This quantity depends only on the mean $\mu_{\bar{\xi}}$ and variance $\sigma_{\bar{\xi}}^2$ of the offspring distribution $\bar{\xi}$. Then, we apply the result that the exploration process is stochastically dominated by a Poisson branching process with $\bar{\xi} = \mathsf{Po}(c)$; in particular, the random variable T_ℓ^2 is stochastically dominated by the random variable $\bar{T}_\ell^2$. (Recall that if Y stochastically dominates X, we have $\mathbb{E}[X] \leq \mathbb{E}[Y]$, see, e.g., [41, Sec. 2.3].)

First, from $\bar{T}_\ell = \bar{Z}_0 + \bar{Z}_1 + \ldots \bar{Z}_\ell$, we obtain

$$\mathbb{E}[\bar{T}_\ell^2] = \sum_{i=0}^{\ell}\mathbb{E}[\bar{Z}_i^2] + 2\sum_{i=1}^{\ell}\sum_{j=0}^{i-1}\mathbb{E}[\bar{Z}_i\bar{Z}_j]. \tag{103}$$

Using the definition of $\bar{Z}_i$ and the law of total expectation, it can be shown that for $i > j$, we have

$$\mathbb{E}[\bar{Z}_i\bar{Z}_j] = \mu_{\bar{\xi}}^{i-j}\mathbb{E}[\bar{Z}_j^2]. \tag{104}$$

Inserting (104) into (103) leads to

$$\mathbb{E}[\bar{T}_\ell^2] = \sum_{i=0}^{\ell}\mathbb{E}[\bar{Z}_i^2] + 2\sum_{i=1}^{\ell}\sum_{j=0}^{i-1}\mu_{\bar{\xi}}^{i-j}\mathbb{E}[\bar{Z}_j^2]. \tag{105}$$

Next, we can use the well-known expressions for the mean and variance of $\bar{Z}_i$ (see, e.g., [42, p. 396]) to obtain

$$\mathbb{E}[\bar{Z}_\ell^2] = \begin{cases} \sigma_{\bar{\xi}}^2\mu_{\bar{\xi}}^{\ell-1}\frac{\mu_{\bar{\xi}}^{\ell}-1}{\mu_{\bar{\xi}}-1} + \mu_{\bar{\xi}}^{2\ell}, & \mu_{\bar{\xi}} \neq 1 \\ \ell\sigma_{\bar{\xi}}^2 + 1, & \mu_{\bar{\xi}} = 1 \end{cases}. \tag{106}$$

Inserting (106) into (105) leads to the desired explicit characterization of $\mathbb{E}[\bar{T}_\ell^2]$. Of particular interest here is the case where the offspring distribution is Poisson with mean c. In this case, we have $\mu_{\bar{\xi}} = c$ and $\sigma_{\bar{\xi}}^2 = c$, which leads to

$$\mathbb{E}[\bar{T}_\ell^2] = \begin{cases} \frac{c^{2\ell+3}-1-(2\ell+3)c^\ell(c-1)}{(c-1)^3}, & c \neq 1 \\ \frac{(\ell+1)(\ell+2)(2\ell+3)}{6}, & c = 1 \end{cases}. \tag{107}$$

Finally, using the same steps as in the proof of [41, Th. 4.2] and [41, Th. 3.20] one can show that the random variable T_ℓ^2 is stochastically dominated by the random variable $\bar{T}_\ell^2$. Hence, (107) is an upper bound on $\mathbb{E}[T_\ell^2]$, i.e., we have $\mathbb{E}[T_\ell^2] \leq \mathbb{E}[\bar{T}_\ell^2]$.

REFERENCES

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

[2] G. D. Forney, Jr., "Concatenated codes," Ph.D. dissertation, Massachusetts Institute of Technology, 1965.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proc. IEEE Int. Conf. Communications (ICC)*, Geneva, Switzerland, 1993.

[4] P. Elias, "Error-free coding," *IRE Trans. Inf. Theory*, vol. 4, no. 4, pp. 29–37, Apr. 1954.

[5] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[6] N. Abramson, "Cascade decoding of cyclic product codes," *IEEE Trans. Commun. Tech.*, vol. 16, no. 3, pp. 398–402, Jun. 1968.

[7] W. Ryan and S. Lin, *Channel Codes Classical and Modern*. Cambridge University Press, 2009.

[8] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *J. Lightw. Technol.*, vol. 30, no. 1, pp. 110–117, Jan. 2012.

[9] J. Justesen and T. Høholdt, "Analysis of iterated hard decision decoding of product codes with Reed-Solomon component codes," in *Proc. IEEE Information Theory Workshop (ITW)*, Tahoe City, CA, 2007.

[10] J. Justesen, K. J. Larsen, and L. A. Pedersen, "Error correcting coding for OTN," *IEEE Commun. Mag.*, vol. 59, no. 9, pp. 70–75, Sep. 2010.

[11] J. Justesen, "Performance of product codes and related structures with iterated decoding," *IEEE Trans. Commun.*, vol. 59, no. 2, pp. 407–415, Feb. 2011.

[12] M. Scholten, T. Coe, and J. Dillard, "Continuously-interleaved BCH (CI-BCH) FEC delivers best in class NECG for 40G and 100G metro applications," in *Proc. Optical Fiber Communication Conf. (OFC)*, San Diego, CA, 2010.

[13] Y.-Y. Jian, H. D. Pfister, K. R. Narayanan, R. Rao, and R. Mazahreh, "Iterative hard-decision decoding of braided BCH codes for high-speed optical communication," in *Proc. IEEE Glob. Communication Conf. (GLOBECOM)*, Atlanta, GA, 2014.

[14] L. M. Zhang and F. R. Kschischang, "Staircase codes with 6% to 33% overhead," *J. Lightw. Technol.*, vol. 32, no. 10, pp. 1999–2002, May 2014.

[15] C. Häger, A. Graell i Amat, H. D. Pfister, A. Alvarado, F. Brännström, and E. Agrell, "On parameter optimization for staircase codes," in *Proc. Optical Fiber Communication Conf. (OFC)*, Los Angeles, CA, 2015.

[16] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

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

[18] R. Gallager, "Low-density parity-check codes," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, 1963.

[19] M. Schwartz, P. Siegel, and A. Vardy, "On the asymptotic performance of iterative decoders for product codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Adelaide, SA, 2005.

[20] S. Hirasawa, M. Kasahara, Y. Sugiyama, and T. Namekawa, "Modified product codes," *IEEE Trans. Inf. Theory*, vol. 30, no. 2, pp. 299–306, Mar. 1984.

[21] M. Alipour, O. Etesami, G. Maatouk, and A. Shokrollahi, "Irregular product codes," in *Proc. IEEE Information Theory Workshop (ITW)*, Lausanne, Switzerland, 2012.

[22] A. J. Feltström, D. Truhachev, M. Lentmaier, and K. S. Zigangirov, "Braided block codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2640–2658, Jul. 2009.

[23] B. Bollobás, S. Janson, and O. Riordan, "The phase transition in inhomogeneous random graphs," *Random Structures and Algorithms*, vol. 31, no. 1, pp. 3–122, Aug. 2007.

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

[25] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proc. 9th Annual ACM-SIAM Symp. Discrete Algorithms*, San Franscisco, CA, 1998.

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

[27] M. Lentmaier, M. B. Tavares, and G. P. Fettweis, "Exact erasure channel density evolution for protograph-based generalized LDPC codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Seoul, South Korea, Jun. 2009.

[28] M. Lentmaier and G. P. Fettweis, "On the thresholds of generalized LDPC convolutional codes based on protographs," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Austin, TX, 2010.

[29] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," *IPN Progress Report 42-154, JPL*, 2005.

[30] N. Miladinovic and M. Fossorier, "Generalized LDPC codes and generalized stopping sets," *IEEE Trans. Commun.*, vol. 56, no. 2, pp. 201–212, Feb. 2008.

[31] Y.-Y. Jian, H. D. Pfister, and K. R. Narayanan, "Approaching capacity at high rates with iterative hard-decision decoding," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Cambridge, MA, 2012.

[32] ——, "Approaching capacity at high-rates with iterative hard-decision decoding," *arxiv:1202.6095v3 [cs.IT]*, May 2015. [Online]. Available: https://arxiv.org/pdf/1202.6095v3

[33] L. M. Zhang, D. Truhachev, and F. R. Kschischang, "Spatially-coupled split-component codes with bounded-distance component decoding," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Hong Kong, 2015.

[34] B. Pittel, J. Spencer, and N. Wormald, "Sudden emergence of a giant k-core in a random graph," *Journal of Combinatorial Theory, Series B*, vol. 67, no. 1, pp. 111–151, May 1996.

[35] P. Erdős and A. Rényi, "On random graphs I," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[36] E. N. Gilbert, "Random graphs," *Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, Dec. 1959.

[37] H. D. Pfister, S. K. Emmadi, and K. Narayanan, "Symmetric product codes," in *Proc. Information Theory and Applications Workshop (ITA)*, San Diego, CA, 2015.

[38] B. Bollobás and Riordan O., "Random graphs and branching processes," in *Handbook of Large-Scale Random Networks*, 2009, pp. 15–115.

[39] O. Riordan, "The k-core and branching processes," *Combinatorics, Probability and Computing*, vol. 17, no. 1, pp. 111–136, Jun. 2007.

[40] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed. Wiley-Interscience, Aug. 2000.

[41] R. van der Hofstad, "Random graphs and complex networks. Vol. I," 2014.

[42] S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes*, 2nd ed. Academic Press, 1975.

[43] A. Dembo and A. Montanari, "Ising models on locally tree-like graphs," *Ann. Appl. Probab.*, vol. 20, no. 2, pp. 565–592, 2010.

[44] A. Montanari, "Statistical mechanics and algorithms on sparse and random graphs," Oct. 2014.

[45] M. G. Luby, M. Mitzenmacher, M. A. Shokrollah, and D. A. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. AMC Symp. on Theory of Computing (STOC)*, New York, USA, 1998.

[46] J. S. Rosenthal, *A First Look At Rigorous Probability Theory*, 2nd ed. World Scientific, 2006.

[47] L. Warnke, "On the method of typical bounded differences," *Combinatorics, Probability and Computing*, vol. 25, no. 2, pp. 269–299, Mar. 2016.

[48] B. P. Smith, "Error-correcting codes for fibre-optic communication systems," Ph.D. dissertation, University of Toronto, 2011.

[49] B. Söderberg, "General formalism for inhomogeneous random graphs," *Phys. Rev. E*, vol. 66, no. 6, Dec. 2002.

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

[51] A. Yedla, Y.-Y. Jian, P. S. Nguyen, and H. D. Pfister, "A simple proof of Maxwell saturation for coupled scalar recursions," *IEEE Trans. Inf. Theory*, vol. 60, no. 11, pp. 6943–6965, Nov. 2014.

[52] C. Häger, H. D. Pfister, A. Graell i Amat, and F. Brännström, "Deterministic and ensemble-based spatially-coupled product codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Barcelona, Spain, 2016.

[53] ——, "Density evolution and error floor analysis of staircase and braided codes," in *Proc. Optical Fiber Communication Conf. (OFC)*, Anaheim, CA, 2016.

[54] C. Häger, A. Graell i Amat, H. D. Pfister, and F. Brännström, "Density evolution for deterministic generalized product codes with higher-order modulation," in *Proc. Int. Symp. Turbo Codes and Iterative Information Processing (ISTC)*, Brest, France, 2016.

[55] M. A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," in *Proc. Int. Symp. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, 1999.

[56] I. Gradshteyn and D. Ryzhik, *Table of Integrals, Series, and Products, Seventh Edition*, 7th ed. Academic Press, 2007.

[57] B. Smith, M. Ardakani, W. Yu, and F. R. Kschischang, "Design of irregular LDPC codes with optimized performance-complexity tradeoff," *IEEE Trans. Commun.*, vol. 58, no. 2, pp. 489–499, Feb. 2010.

[58] M. Molloy, "Cores in random hypergraphs and boolean formulas," *Random Structures and Algorithms*, vol. 27, no. 1, pp. 124–135, Aug. 2005.

[59] A. Amraoui, A. Montanari, T. J. Richardson, and R. L. Urbanke, "Finite-length scaling for iteratively decoded LDPC ensembles," *IEEE Trans. Inf. Theory*, vol. 55, no. 2, pp. 473–498, Feb. 2009.

[60] S. Janson and M. J. Luczak, "Asymptotic normality of the k-core in random graphs," *Ann. Appl. Probab.*, vol. 18, no. 3, pp. 1085–1137, Jun. 2008.