Efficient Encoding of Low-Density Parity-Check Codes*

Tom Richardson † Rüdiger Urbanke ‡

March 6, 2001

Abstract

Low-density parity-check codes can be considered serious competitors to turbo codes in terms of performance and complexity and they are based on a similar philosophy: constrained random code ensembles and iterative decoding algorithms.

In this paper we consider the encoding problem for low-density parity-check codes. More generally, we consider the *encoding* problem for codes specified by sparse parity-check matrices. We show how to exploit the sparseness of the parity-check matrix to obtain efficient encoders. For the (3,6)-regular low-density parity-check code for example, the complexity of encoding is essentially quadratic in the block length. However, we show that the associated coefficient can be made quite small, so that encoding codes even of length $n \simeq 100,000$ is still quite practical. More importantly, we will show that "optimized" codes actually admit linear time encoding.

Keywords: turbo codes, parity-check, decoding, encoding, sparse matrices, random graphs, binary erasure channel.

1 Introduction

Low-density parity-check (LDPC) codes were originally invented and investigated by Gallager [1]. The crucial innovation was Gallager's introduction of iterative decoding algorithms (or message-passing decoders) which he showed to be capable of achieving a significant fraction of channel capacity at low complexity. Except for the papers by Zyablov and Pinsker [2], Margulis [3] and Tanner [4] the field then lay dormant for the next 30 years. Interest in LDPC codes was rekindled in the wake of the discovery of turbo-codes and LDPC codes were independently rediscovered by both MacKay and Neal [5] and Wiberg [6].[1] The past few years have brought many new developments in this area. First, in several papers Luby, Mitzenmacher, Shokrollahi, Spielman and Stemann

*This work was performed while both authors were at Bell Labs.

†Flarion Technologies, Bedminster, NJ 07921, USA, email: richardson@flarion.com

‡EPFL, LTHC-DSC, CH-1015 Lausanne, email: Rudiger.Urbanke@epfl.ch

[1]Similar concepts have also appeared in the physics literature [7, 8].

introduced new tools for the investigation of message-passing decoders for the binary erasure channel (BEC) and the binary symmetric channel (BSC) (under hard decision message-passing decoding) [9, 10], and they extended Gallager's definition of LDPC codes to include *irregular* codes, see also [5]. The same authors also exhibited sequences of codes which, asymptotically in the block-length, provably achieve capacity on a BEC. It was then shown in [11] that similar analytic tools can be used to study the asymptotic behavior of a very broad class of message passing algorithms for a wide class of channels and it was demonstrated in [12] that LDPC codes can come extremely close to capacity on many channels.

In many ways LDPC codes can be considered serious competitors to turbo codes. In particular, LDPC codes exhibit an asymptotically better performance than turbo codes and they admit a wide range of trade-offs between performance and decoding complexity. One major criticism concerning LDPC codes has been their apparent high *encoding* complexity. Whereas turbo codes can be encoded in linear time, a straightforward encoder implementation for a LDPC code has complexity quadratic in the block length. Several authors have addressed this issue:

1. It was suggested in [13] and [9] to use cascaded rather than bipartite graphs. By choosing the number of stages and the relative size of each stage carefully one can construct codes which are encodable and decodable in linear time. One drawback of this approach lies in the fact that each stage (which acts like a subcode) has a length which is in general considerably smaller than the length of the overall code. This results, in general, in a performance loss compared to a standard LDPC code with the same overall length.

2. In [14] it was suggested to force the parity check matrix to have (almost) lower triangular form, i.e., the ensemble of codes is restricted not only by the degree constraints but also by the constraint that the parity check matrix have lower triangular shape. This restriction guarantees a linear time encoding complexity but, in general, it also results in some loss of performance.

It is the aim of this paper to show that, even without cascade constructions or restrictions on the shape of the parity check matrix, the encoding complexity is quite manageable in most cases and provably linear in many cases. More precisely, for a $(3, 6)$-regular code of length n the encoding complexity seems indeed to be of order n^2 but the actual number of operations required is no more than $0.017^2 n^2 + O(n)$, and, because of the extremely small constant factor, even large block lengths admit practically feasible encoders. We will also show that "optimized" irregular codes have a *linear* encoding complexity and that the required amount of preprocessing is of order at most $n^{3/2}$.

The proof of these facts is achieved in several stages. We first show in Section 2 that the encoding complexity is upper bounded by $n + g^2$, where g, the *gap*, measures in some way to be made precise shortly, the "distance" of the given parity check matrix to a lower triangular matrix. In Section 3 we then discuss several greedy algorithms to triangulate matrices and we show that for these algorithms, when applied to elements of a given ensemble, the gap concentrates around its expected value with high probability. As mentioned above, for the $(3,6)$-regular code the best greedy algorithm which we discuss results in an expected gap of $0.017n$. Finally, in Section 4 we prove that for all known "optimized" codes the expected gap is actually of order less than $\sqrt{n}$, resulting in the promised linear encoding complexity. In practice the gap is usually a small constant. The $\sqrt{n}$ bound can be improved but it would require a significantly more complex presentation.

We finish this section with a brief review of some basic notation and properties concerning LDPC codes. For a more thorough discussion we refer the reader to [1, 11, 12].

Low-density parity-check codes are linear codes. Hence, they can be expressed as the null space of a *parity check* matrix H, i.e., x is a codeword if and only if

$$Hx^T = 0^T.$$

The modifier 'low-density' applies to H; The matrix H should be sparse. For example, if H has dimension $\frac{n}{2} \times n$, where n is even, then we might require H to have 3 ones per column and 6 ones per row. Conditioned on these constraints we choose H at random as discussed in more detail below. We refer to the associated code as a $(3,6)$-regular LDPC code. The sparseness of H enables efficient (sub-optimal) decoding, while the randomness ensures (in the probabilistic sense) a good code [1].

Example 1. [Parity-Check Matrix of $(3,6)$-Regular Code of Length 12] The following matrix H will serve as an example.

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \tag{1}$$

□

In the theory of LDPC codes it is customary and useful not to focus on particular codes but to consider ensembles of codes. These ensembles are usually defined in terms of ensembles of *bipartite graphs* [13, 15]. E.g., the bipartite graph which represents the code defined in Example 1 is shown in Fig. 1. The *left* set of nodes represents the *variables* whereas the *right* set of nodes represents the *constraints*. An ensemble of bipartite graphs is defined in terms of a pair of *degree distributions*. A degree distribution $\gamma(x) = \sum_i \gamma_i x^{i-1}$ is simply a polynomial with non-negative real coefficients satisfying $\gamma(1) = 1$. Typically, γ_i denotes the fraction of edges in a graph which are incident to a node (variable or constraint node as the case may be) of degree i. In the sequel, we will use the shorthand $\int\gamma$ to denote $\sum_{i\geq 1}\frac{\gamma_i}{i} = \int_0^1 \gamma(x)\,dx$. This quantity gives the inverse of the average node degree. Associated to a degree distribution pair (λ, ρ) is the *rate* $r(\lambda, \rho)$ defined as

$$r(\lambda, \rho) := 1 - \frac{\int\rho}{\int\lambda}. \tag{2}$$

E.g., for the degree distribution pair (x^2, x^5), which corresponds to the regular $(3, 6)$ Gallager code, the rate is $\frac{1}{2}$.

Given a pair (λ, ρ) of degree distributions and a natural number n, we define an *ensemble* of bipartite graphs $\mathcal{C}^n(\lambda, \rho)$ in the following way. All graphs in the ensemble $\mathcal{C}^n(\lambda, \rho)$ will have *left* nodes which are associated to λ and *right* nodes which are associated to ρ. More precisely, assume that $\lambda(x) := \sum_{i\geq 1}\lambda_i x^{i-1}$ and $\rho(x) := \sum_{i\geq 1}\rho_i x^{i-1}$. We can convert these degree distributions into *node perspective* by defining $\tilde{\lambda}_i := \frac{\lambda_i}{i\int\lambda}$ and $\tilde{\rho}_i := \frac{\rho_i}{i\int\rho}$. Each graph in $\mathcal{C}^n(\lambda, \rho)$ has $n\tilde{\lambda}_i$ left nodes of degree i and $(1 - r(\lambda, \rho))n\tilde{\rho}_i$ right nodes of degree i. The order of these nodes is arbitrary but fixed. Here, to simplify notation, we assume that (λ, ρ) and n are chosen in such a way that all these quantities are integers. A node of degree i has i *sockets* from which the i edges emanate and these sockets are *ordered*. So in total there are

$$s := \sum_{i\geq 1} in\tilde{\lambda}_i = \sum_{i\geq 1} n\frac{\lambda_i}{\int\lambda} = \frac{n}{\int\lambda} = \frac{(1 - r(\lambda, \rho))n}{\int\rho} = (1 - r(\lambda, \rho))\sum_{i\geq 1} n\frac{\rho_i}{\int\rho} = (1 - r(\lambda, \rho))\sum_{i\geq 1} in\tilde{\rho}_i$$

ordered sockets on the left as well as on the right. Let σ be a permutation on $[s] := \{1, \cdots, s\}$. We can associate a graph to such a permutation by connecting the i-th socket on the left to the $\sigma(i)$-th socket on the right. Letting σ run over the set of permutations on $[s]$ generates a set of graphs. Endowed with the uniform probability distribution this is the ensemble $\mathcal{C}^n(\lambda, \rho)$. Therefore, if in the future we choose a graph at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ then the underlying probability distribution is the uniform one.

It remains to associate a code to every element of $\mathcal{C}^n(\lambda, \rho)$. We will do so by associating a parity check matrix to each graph. At first glance it seems natural to define the parity check matrix associated to a given element in $\mathcal{C}^n(\lambda, \rho)$ as the $\{0,1\}$-matrix which has a non-zero entry at row i and column j iff the i-th right node is connected to the j-th left node. Unfortunately the possible presence of multiple edges between pairs of nodes requires a more careful definition. Since the encoding is done over the field GF(2), we define the parity check matrix H as that $\{0,1\}$-matrix which has a non-zero entry at row i and column j iff the i-th right node is connected to the j-th left node an *odd* number of times. As we will see the encoding is accomplished in two steps, a *preprocessing* step, which is an offline calculation performed once only for the given code, and the actual encoding step which is the only data dependent part. For the preprocessing step it is more natural to work with matrices which contain the multiplicities of edges and therefore we define the *extended parity check matrix* $\hat{H}$ as that matrix which has an entry d at row i and column j iff the i-th right node is connected to the j-th left node by d edges. Clearly, H is equal to $\hat{H}$ modulo 2. In the sequel we will also refer to these two matrices as the *adjacency matrix* and the *extended adjacency matrix* of the bipartite graph. Since for every graph there is an associated code, we will use these two terms interchangeably so we will, e.g., refer to codes as elements of $\mathcal{C}^n(\lambda, \rho)$.

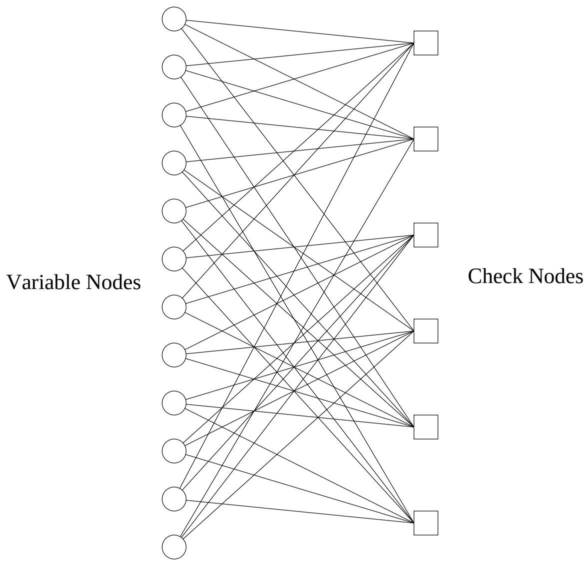

Figure 1: Graphical representation of a $(3, 6)$-regular LDPC code of length 12. The left nodes represent the variable nodes whereas the right nodes represent the check nodes.

Most often LDPC codes are used in conjunction with *message-passing decoders*. Recall that there is a received message associated to each variable node which is the result of passing the corresponding bit of the codeword through the given channel. The decoding algorithm proceeds in *rounds*. At each round a message is sent from each variable node to each neighboring check node, indicating some estimate of the associated bit's value. In turn, each check node collects its incoming messages and, based on this information, sends messages back to the incident variable nodes. Care must be taken to send out only *extrinsic* information, i.e., the outgoing message along a given edge must not depend on the incoming message along the same edge. As we will see, the preprocessing step for the encoding is closely related to the message-passing decoder for the BEC. We will therefore review this particular decoder in more detail.

Assume we are given a code in $\mathcal{C}^n(\lambda, \rho)$ and assume that we use this code to transmit over a BEC with an erasure probability of α. An asymptotic expected fraction α of the variable nodes will be *erasures* and the remaining fraction $(1-\alpha)$ will be *known*. We first formulate the iterative decoder not as a message-passing decoder but in a language which is more suitable for our current purpose, see [9]:

Decoder for the Binary Erasure Channel:

0. [Initialization]

1. [Stop or Extend] If there is no known variable node and no check node of degree one then output the (partial) codeword and stop. Otherwise, all known variable nodes and all their adjacent edges are deleted.

2. [Declare Variables as Known] Any variable node which is connected to a degree one check node is declared to be known. Goto 1.

This decoder can equivalently be formulated as a message-passing decoder. Messages are from the set $\{0, 1\}$ with a 0 indicating that the corresponding bit has not been determined yet (along the given edge). We will call a 0 message an *erasure message*. At a variable node the outgoing message along an edge $\mathtt{e}$ is the erasure message if the received message associated to this node is an erasure and if all incoming messages (excluding the incoming message along edge $\mathtt{e}$) are erasure messages, otherwise the outgoing message is a 1. At a check node the outgoing message along an edge $\mathtt{e}$ is the erasure message if at least one of the incoming messages (excluding the incoming message along edge $\mathtt{e}$) is the erasure message, and a 1 otherwise. If we declare that an originally erased variable node becomes *known* as soon as it has at least one incoming message which is not an erasure then one can check that at any time the set of known variable nodes is indeed identical under both descriptions.

It was shown in [16] that (asymptotically in n) the expected fraction α_ℓ of erasure messages after the ℓ-th decoding round is given by

$$\alpha_\ell = \alpha\lambda(1-\rho(1-\alpha_{\ell-1})), \tag{3}$$

where $\alpha_0 = \alpha$. Let $\alpha^*(\lambda,\rho)$, called the *threshold* of the degree distribution pair, be defined as

$$\alpha^*(\lambda,\rho) := \sup\{0 \le \alpha \le 1 : \alpha_\ell(\alpha) \stackrel{\ell\to\infty}{\longrightarrow} 0 \text{ where } \alpha_\ell(\alpha) := \alpha\lambda(1-\rho(1-\alpha_{\ell-1}));\ \alpha_0=\alpha\}. \tag{4}$$

Note first that the function $f(x,y) := y\lambda(1-\rho(1-x))$ is increasing in both its arguments for $x,y \in [0,1]$. It follows by finite induction that if $\alpha_\ell(\alpha) \stackrel{\ell\to\infty}{\to} 0$ then $\alpha_\ell(\alpha') \stackrel{\ell\to\infty}{\to} 0$ for any $\alpha' \le \alpha$. If we choose $\alpha < \alpha^*(\lambda,\rho)$, then the expected fraction of erasure messages converges to zero. Consequently, the decoder will be successful with high probability in this case. If, on the other hand, we choose $\alpha > \alpha^*(\lambda,\rho)$ then with high probability the decoding process will not succeed. We will see shortly that, correctly interpreted, this decoding procedure constitutes the basis for all preprocessing algorithms that we consider in this paper.

Example 2. [(3,6)-regular code] Let $(\lambda(x),\rho(x)) = (x^2,x^5)$. Then $r(x^2,x^5) = \frac{1}{2}$. The exact threshold $\alpha^*(x^2,x^5)$ was determined in [17] and can be expressed as follows: Let σ be given by

$$\sigma = \frac{1}{36} + \frac{\sqrt{\frac{25}{324}-a+b}}{2} + \frac{\sqrt{\frac{25}{162}+a-b+\frac{685}{2916\sqrt{\frac{25}{324}-a+b}}}}{2},$$

where $a = \frac{22}{27}5^{\frac{2}{3}}\left(\frac{2}{-85+3\sqrt{24465}}\right)^{\frac{1}{3}}$ and $b = \frac{1}{27}\left(\frac{5}{2}\left(-85+3\sqrt{24465}\right)\right)^{\frac{1}{3}}$. Then $\alpha^*(x^2,x^5) := \frac{1-\sigma}{(1-\sigma^5)^2} \sim 0.42944$. $\square$

2 Efficient Encoders Based on Approximate Lower Triangulations

In this section we shall develop an algorithm for constructing efficient encoders for LDPC codes. The efficiency of the encoder arises from the sparseness of the parity check matrix H and the algorithm can be applied to any (sparse) H. Although our example is binary, the algorithm applies generally to matrices H whose entries belong to a field F. We assume throughout that the rows of H are linearly independent. If the rows are linearly dependent, then the algorithm which constructs the encoder will detect the dependency and either one can choose a different matrix H or one can eliminate the redundant rows from H in the encoding process.

Assume we are given an $m \times n$ parity-check matrix H over F. By definition the associated code consists of the set of n-tuples x over F such that

$$Hx^T = 0^T.$$

Probably the most straightforward way of constructing an encoder for such a code is the following. By means of Gaussian elimination bring H into an equivalent lower triangular form as shown in Fig. 2. Split the vector x into a *systematic* part s, $s \in \mathrm{F}^{n-m}$, and a *parity* part p, $p \in \mathrm{F}^m$, such that

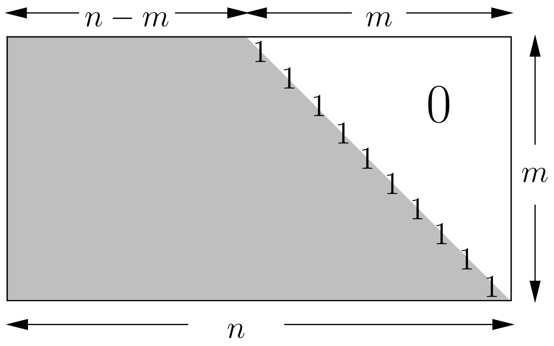

Figure 2: An equivalent parity-check matrix in lower triangular form.

$x = (s, p)$. Construct a *systematic* encoder as follows: (i) Fill s with the $(n-m)$ desired information symbols. (ii) Determine the m parity-check symbols using *back-substitution*. More precisely, for $l \in [m]$ calculate

$$p_l = \sum_{j=1}^{n-m} H_{l,j} s_j + \sum_{j=1}^{l-1} H_{l,j+k} p_j.$$

What is the complexity of such an encoding scheme? Bringing the matrix H into the desired form requires $O(n^3)$ operations of *preprocessing*. The actual encoding then requires $O(n^2)$ operations since, in general, after the preprocessing the matrix will no longer be sparse. More precisely, we expect that we need about $n^2 \frac{r(1-r)}{2}$ `XOR` operations to accomplish this encoding, where r is the rate of the code.

Given that the original parity-check matrix H is sparse one might wonder if encoding can be accomplished in $O(n)$. As we will show, for codes which allow transmission at rates close to capacity, linear time encoding is indeed possible. And for those codes for which our encoding scheme still leads to quadratic encoding complexity the constant factor in front of the n^2 term is typically very small so that the encoding complexity stays manageable up to very large blocklengths.

Our proposed encoder is motivated by the above example. Assume that by *performing row and column permutations only* we can bring the parity check matrix into the form indicated in Fig. 3. We say that H is in *approximate lower triangular form*. Note that since this transformation was

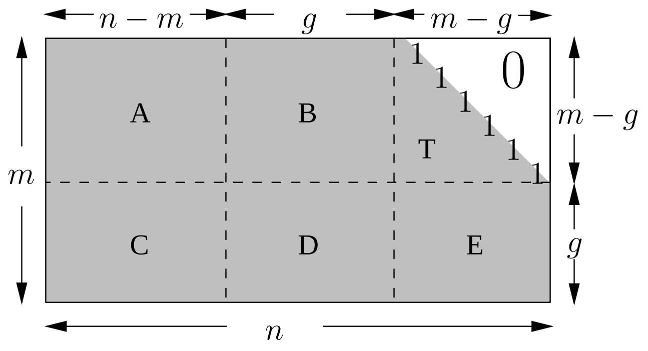

Figure 3: The parity-check matrix in approximate lower triangular form.

accomplished solely by permutations, the matrix is still sparse. More precisely, assume that we bring the matrix in the form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}, \tag{5}$$

where A is $(m-g)\times(n-m)$, B is $(m-g)\times g$, T is $(m-g)\times(m-g)$, C is $g\times(n-m)$, D is $g\times g$ and finally E is $g\times(m-g)$. Further all these matrices are sparse[2] and T is lower triangular with ones along the diagonal. Multiplying this matrix from the left by

$$\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix}, \tag{6}$$

we get

$$\begin{pmatrix} A & B & T \\ -ET^{-1}A+C & -ET^{-1}B+D & 0 \end{pmatrix}. \tag{7}$$

Let $x=(s,p_1,p_2)$ where s denotes the systematic part, p_1 and p_2 combined denote the parity part, p_1 has length g and p_2 has length $(m-g)$. The defining equation $Hx^T=0^T$ splits naturally into two equations, namely

$$\begin{aligned} As^T + Bp_1^T + Tp_2^T &= 0, \quad \text{and} \qquad (8)\\ (-ET^{-1}A+C)s^T + (-ET^{-1}B+D)p_1^T &= 0. \qquad (9)\end{aligned}$$

[2]More precisely, each matrix contains at most $O(n)$ elements.

Operation	Comment	Complexity
As^T	Multiplication by sparse matrix	O(n)
$T^{-1}\left[As^T\right]$	$T^{-1}\left[As^T\right] = y^T \Leftrightarrow \left[As^T\right] = Ty^T$	O(n)
$-E\left[T^{-1}As^T\right]$	Multiplication by sparse matrix	O(n)
Cs^T	Multiplication by sparse matrix	O(n)
$\left[-ET^{-1}As^T\right] + \left[Cs^T\right]$	Addition	O(n)
$-\phi^{-1}\left[-ET^{-1}As^T + Cs^T\right]$	Multiplication by dense $g \times g$ matrix	O(g^2)

Table 1: Efficient computation of $p_1^T = -\phi^{-1}(-ET^{-1}A + C)s^T$.

Operation	Comment	Complexity
As^T	Multiplication by sparse matrix	O(n)
Bp_1^T	Multiplication by sparse matrix	O(n)
$\left[As^T\right] + \left[Bp_1^T\right]$	Addition	O(n)
$-T^{-1}\left[As^T + Bp_1^T\right]$	$-T^{-1}\left[As^T + Bp_1^T\right] = y^T \Leftrightarrow -\left[As^T + Bp_1^T\right] = Ty^T$	O(n)

Table 2: Efficient computation of $p_2^T = -T^{-1}(As^T + Bp_1^T)$.

Define $\phi := -ET^{-1}B + D$ and assume for the moment that ϕ is non-singular. We will discuss the general case shortly. Then from equation (9) we conclude that

$$p_1^T = -\phi^{-1}(-ET^{-1}A + C)s^T.$$

Hence, once the $g \times (n-m)$ matrix $-\phi^{-1}(-ET^{-1}A+C)$ has been precomputed the determination of p_1 can be accomplished in complexity $O(g \times (n-m))$ simply by performing a multiplication with this (generically dense) matrix. This complexity can be further reduced as shown in Table 1. Rather than precomputing $-\phi^{-1}(-ET^{-1}A + C)$ and then multiplying with s^T we can determine p_1 by breaking the computation into several smaller steps, each of which is efficiently computable.

To this end, we first determine As^T, which has complexity $O(n)$ since A is sparse. Next, we multiply the result by T^{-1}. Since $T^{-1}[As^T] = y^T$ is equivalent to the system $[As^T] = Ty^T$ this can also be accomplished in $O(n)$ by back-substitution, since T is lower triangular and also sparse. The remaining steps are fairly straightforward. It follows that the overall complexity of determining p_1 is $O(n + g^2)$. In a similar manner, noting from equation (8) that $p_2^T = -T^{-1}(As^T + Bp_1^T)$, we can accomplish the determination of p_2 in complexity $O(n)$ as shown step by step in Table 2.

A summary of the proposed encoding procedure is given in Table 3. It entails two steps: A *preprocessing* step and the actual *encoding* step. In the preprocessing step we first perform row and column permutations to bring the parity check matrix into approximate lower triangular form

Preprocessing: Input: Non-singular parity-check matrix H. Output: An equivalent parity check matrix of the form $\begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$ such that $-ET^{-1}B + D$ is non-singular.

1. [Triangulation] Perform row and column permutations to bring the parity check matrix H into approximate lower triangular form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$$

with as small a gap g as possible. We will see in subsequent sections how this can be accomplished efficiently.

2. [Check Rank] Use Gaussian elimination to effectively perform the pre-multiplication

$$\begin{pmatrix} I & 0 \\ ET^{-1} & I \end{pmatrix} \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} = \begin{pmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{pmatrix}, \tag{10}$$

in order to check that $-ET^{-1}B + D$ is non-singular, performing further column permutations if necessary to ensure this property. (Singularity of H can be detected at this point.)

Encoding: Input: Parity-check matrix of the form $\begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$ such that $-ET^{-1}B + D$ is non-singular and a vector $s \in \mathrm{F}^{n-m}$. Output: The vector $x = (s, p_1, p_2)$, $p_1 \in \mathrm{F}^g$, $p_2 \in \mathrm{F}^{m-g}$, such that $Hx^T = 0^T$.

1. Determine p_1 as shown in Table 1.

2. Determine p_2 as shown in Table 2.

Table 3: Summary of the proposed encoding procedure. It entails two steps: A preprocessing step and the actual encoding step.

with as small a gap g as possible. We will see in subsequent sections how this can be accomplished efficiently. We also need to check whether $\phi := -ET^{-1}B + D$ is non-singular. Rather than pre-multiplying by the matrix $\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix}$ this task can be accomplished efficiently by Gaussian elimination. If, after clearing the matrix E the resulting matrix ϕ is seen to be singular we can simply perform further column permutations to remove this singularity. This is always possible when H is not rank deficient, as assumed. The actual encoding then entails the steps listed in Tables 1 and 2.

We will now demonstrate this procedure by means of our running example.

Example 3. [Parity Check Matrix of $(3, 6)$-regular code of length 12] For this example if we simply reorder the columns such that, according to the original order, we have the ordering 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 8, 9, then we put the parity check matrix into an approximate lower triangular form with $g = 2$,

$$\left(\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right) = \left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \mathbf{1} \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right). \tag{11}$$

We now use Gaussian elimination to clear E. This results in

$$\left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \mathbf{1} \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}\right).$$

We see that $\phi := -ET^{-1}B + D = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ is singular. This singularity can be removed if we exchange e.g. column 5 with column 8 which gives $\phi = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. In terms of the original order the final column order is then 1, 2, 3, 4, 10, 6, 7, 5, 11, 12, 8, 9 and the resulting equivalent parity check matrix is

$$\left(\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right) = \left(\begin{array}{cccccc|cc|cccc} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \mathbf{1} & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{1} \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right). \tag{12}$$

Assume now we choose $s = (1, 0, 0, 0, 0, 0)$. To determine p_1 we follow the steps listed in Table 1. We get $As^T = (1, 1, 0, 1)^T$, $T^{-1}[As^T] = (1, 1, 0, 0)^T$, $-E[T^{-1}As^T] = (0, 1)^T$, $Cs^T = (0, 0)^T$, $[-ET^{-1}As^T] + [Cs^T] = (0, 1)^T$, and $\phi^{-1}[-ET^{-1}As^T + Cs^T] = (0, 1)^T = p_1^T$. In a similar manner we execute the steps listed in Table 2 to determine p_2. We get $Bp_1^T = (0, 1, 0, 0)^T$, $[As^T] + [Bp_1^T] = (1, 0, 0, 1)^T$ and $T^{-1}[As^T + Bp_1^T] = (1, 0, 1, 0)^T = p_2^T$. Therefore the codeword is equal to $(s, p_1, p_2) = (1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0)$. A quick check verifies that $Hx^T = 0^T$, as required. $\square$

3 Approximate Upper-Triangulation Via Greedy Algorithms

We saw in the previous section that the encoding complexity is of order $n + g^2$, where g is the gap of the approximate triangulation. Hence, for a given parity check matrix we are interested in finding an approximate lower triangulation with as small a gap as possible. Given that we are interested in large block-lengths, there is little hope of finding the optimal row and column permutation which results in the minimum gap. So we will limit ourselves to *greedy* algorithms. As discussed in the previous section, the following greedy algorithms work on the extended adjacency matrices since these are, except for the ordering of the sockets, in one-to-one correspondence with the underlying graphs.

To describe the algorithms we first need to extend some of our previous definitions. Recall that for a given pair (μ, ν) of degree distributions we associate to it two important parameters. The first parameter $r(\mu, \nu)$ is the *rate* of the degree distribution pair and is defined in (2). Note that

$$1 - r(\mu, \nu) = \frac{1}{1 - r(\nu, \mu)}. \tag{13}$$

The second parameter $\alpha^*(\mu, \nu)$ is called the *threshold* of the degree distribution pair and is defined in (4). If $r(\mu, \nu) \geq 0$, as we have tacitly assumed so far, then we can think of (μ, ν) as the degree distribution pair of an ensemble of LDPC codes of rate $r(\mu, \nu)$. Further, as discussed in the introduction, in this case it was shown in [9] that $\alpha^*(\mu, \nu)$ is the threshold of this ensemble when transmitting over the BEC assuming a belief propagation decoder. In general, $r(\mu, \nu)$ may be negative and, hence, the degree distribution pair does not correspond to an ensemble of LDPC codes. Nevertheless, the definitions are still meaningful.

Example 4. Let $(\mu(x), \nu(x)) = (x^5, x^2)$. In this case we have $r(x^5, x^2) = -1$ and using the techniques described in [17] the threshold can be determined to be $\alpha^*(x^5, x^2) = \frac{3^{18}}{2^{17}5^5} \sim 0.945851$. $\square$

In a similar way, the definition of the ensemble $\mathcal{C}^l(\mu, \nu)$ as well as the association of (extended) adjacency matrices to elements of $\mathcal{C}^l(\mu, \nu)$ carry over to the case $r(\mu, \nu) \leq 0$. Assume now that, for a given ensemble $\mathcal{C}^l(\mu, \nu)$, we create a new ensemble by simply exchanging the roles of left and right nodes. This new ensemble is equivalent to the ensemble $\mathcal{C}^{(1-r(\mu,\nu))l}(\nu, \mu) = \mathcal{C}^{\frac{1}{1-r(\nu,\mu)}l}(\nu, \mu)$ where we have used (13). For the associated (extended) adjacency matrices this simply amounts to transposition.

Assume we are given a matrix A of dimension $(1 - r)l \times l$ with elements in $\mathbb{N}$, where r is some real valued parameter with $r \leq 1$. We will say that a row and a column are *connected* if the corresponding entry in A is non-zero. Furthermore, we will say that a row (column) has *degree* i if its row (column) sum equals i. Assume now that we want to bring A into approximate lower triangular form. The class of greedy algorithms that we will consider is based on the following simple procedure. Given the $(1 - r)l \times l$ matrix A and a fixed integer k, $k \leq (1 - r)l$, permute, if possible, the rows and columns in such a way that the first row has its last non-zero entry at position $(l-k+1)$. If this first step was successful then fix the first row and permute, if possible, the remaining rows and all columns in such a way that the second row has its last non-zero entry at position $(l-k+2)$. In general, assuming that the first $i-1$ steps were successful, permute at the i-th step, if possible, the last $(1-r)l-(i-1)$ rows and all columns in such a way that the i-th row has its last non-zero entry at position $(l-k+i)$. If this procedure does not terminate before the k-th step then we accomplished an approximate lower triangulation of the matrix A. We will say that A is in approximate lower triangular form with *row gap* $(1-r)l-k$ and *column gap* $l-k$, as shown in Fig. 4.

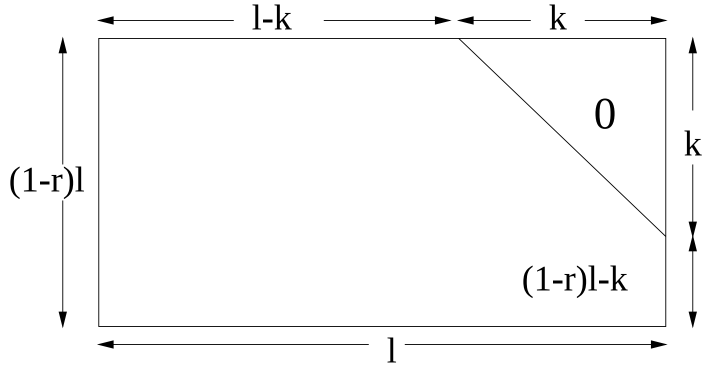

Figure 4: Approximate lower triangulation of the matrix A with row gap $(1-r)l-k$ and column gap $l-k$ achieved by a greedy algorithm.

3.1 Greedy Algorithm A

We will now give a precise description of the greedy algorithm A. The core of the algorithm is the *diagonal extension step.*

Diagonal Extension Step. Assume we are given a matrix A and a subset of the columns which are classified as *known.* In all cases of interest to us either none of these known columns are connected to rows of degree one or all of them are. Assume the latter case. Let $\mathtt{c}_1, \ldots, \mathtt{c}_k$ denote the known columns and let $\mathtt{r}_1, \ldots, \mathtt{r}_k$ be degree one rows such that $\mathtt{c}_i$ is connected to $\mathtt{r}_i$.[3] Reorder, if necessary, the rows and columns of A such that $\mathtt{r}_1, \ldots, \mathtt{r}_k$ form the leading k rows of A and such that $\mathtt{c}_1, \ldots, \mathtt{c}_k$ form the leading k columns of A as shown in Fig. 5, where $\tilde{A}$ denotes the sub-matrix of A which results from deleting the rows and columns indexed by $\mathtt{r}_1, \ldots, \mathtt{r}_k$ and $\mathtt{c}_1, \ldots, \mathtt{c}_k$. Note

[3] $\mathtt{r}_i$ may not be determined uniquely.

that after this reordering the top-left $k \times k$ sub-matrix of A has diagonal form and that the top k rows of A have only this one non-zero entry.

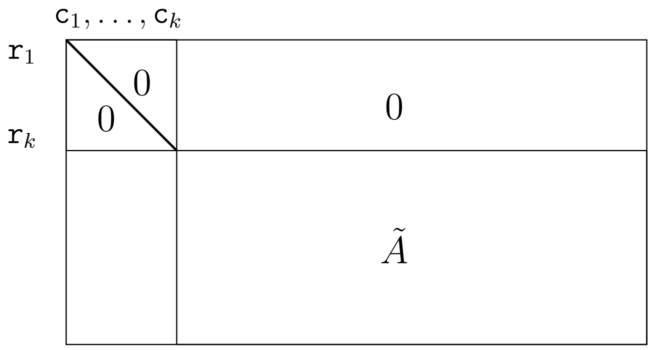

Figure 5: Given the matrix A let $\mathtt{c}_1, \ldots, \mathtt{c}_k$ denote those columns which are connected to rows of degree one and let $\mathtt{r}_1, \ldots, \mathtt{r}_k$ be degree one rows such that $\mathtt{c}_i$ is connected to $\mathtt{r}_i$. Reorder the rows and columns in such a way such that $\mathtt{r}_1, \ldots, \mathtt{r}_k$ form the first k rows and such that $\mathtt{c}_1, \ldots, \mathtt{c}_k$ form the first k columns. Note that the top-left $k \times k$ sub-matrix has diagonal form and that the first k rows have only this one non-zero entry.

By a diagonal extension step we will mean the following: As input we are given the matrix A and a set of known columns. The algorithm performs some row and column permutations and specifies a *residual matrix* $\tilde{A}$. More precisely, to implement the algorithm one does the following. If none of the known columns are connected to rows of degree one then perform a column permutation so that all the known columns form the leading columns of the matrix A. Furthermore, delete these known columns from the original matrix and declare the resulting matrix to be $\tilde{A}$. If, on the other hand, all known columns are connected to rows of degree one then perform a row and column permutation to bring A into the form depicted in Fig. 5. Furthermore, delete the known columns $\mathtt{c}_1, \ldots, \mathtt{c}_k$ and the rows $\mathtt{r}_1, \ldots, \mathtt{r}_k$ from the original matrix and declare the resulting matrix to be $\tilde{A}$.

In terms of this diagonal extension step, greedy algorithm A has a fairly succinct description.

Greedy Algorithm A:

0. [Initialization] Given a matrix A declare each column independently to be *known* with probability $1 - \alpha$ or otherwise to be an *erasure*. Let $\tilde{A} := A$.

1. [Stop or Extend] If $\tilde{A}$ contains neither a known column nor a row of degree one then output the present matrix. Otherwise perform a diagonal extension step.

2. [Declare Variables as Known] Any column in $\tilde{A}$ which is connected to a degree one row is declared to be known. Goto 1.

To see that greedy algorithm A indeed gives rise to an approximate triangulation assume that we start with the $(1-r)l \times l$ matrix A as shown in Fig. 6(a). In the initialization step an expected fraction $(1-\alpha)$ of all columns are classified as known and the rest is classified as erasures. The first time the algorithm performs step one these $(1-\alpha)l$ known columns are reordered to form the leading columns of the matrix A as shown in Fig. 6(b). Assuming that the residual matrix has rows of degree one, the columns connected to these degree one rows are identified in the second step. Let these columns be $\mathtt{c}_1, \ldots, \mathtt{c}_k$ and let $\mathtt{r}_1, \ldots, \mathtt{r}_k$ be degree one rows such that $\mathtt{c}_i$ is connected to $\mathtt{r}_i$. During the second application of step one these new known columns and their associated rows are ordered along a diagonal as shown in Fig. 6(c). Furthermore, in each additional iteration this diagonal is extended further. If this procedure does not stop prematurely then the resulting diagonal has expected length αl and therefore the row gap has expected size $(1-r-\alpha)l$ and the column gap has expected size $(1-\alpha)l$ as shown in Fig. 6(d). If on the other hand the procedure terminates before all columns are exhausted then we get an approximate triangulation by simply reordering the remaining columns to the left. Assuming that the remaining fraction of columns is equal to ϵl then the resulting expected row gap is equal to $(1-r-\alpha+\epsilon)l$ and the resulting expected column gap is equal to $(1-\alpha+\epsilon)l$.

Lemma 1. [Performance of Greedy Algorithm A] Let (μ, ν) be a given degree pair and choose $\alpha < \alpha^*(\mu, \nu)$. Pick a graph at random from the ensemble $\mathcal{C}^l(\mu, \nu)$ and let $\hat{A}$ be its extended adjacency matrix. Apply greedy algorithm A to the extended adjacency matrix $\hat{A}$. Then (asymptotically in l) the row gap is concentrated around the value $(1-r-\alpha)l$ and the column gap is concentrated around the value $(1-\alpha)l$. Letting $\alpha \uparrow \alpha^*$, we see that the minimum row gap achievable with greedy algorithm A is equal to $(1-r-\alpha^*)l$ and that the minimum column gap is equal to $(1-\alpha^*)l$.

Proof. Assume we are given a graph and an associated extended adjacency matrix $\hat{A}$ from the ensemble $\mathcal{C}^l(\mu, \nu)$. Assume first that $r(\lambda, \rho) \geq 0$ so that $\mathcal{C}^l(\mu, \nu)$ represents an ensemble of LDPC codes of rate $r(\lambda, \rho)$. For the same code/graph consider the process of transmission over an erasure channel with erasure probability α followed by decoding using the message passing decoder described in the introduction. Compare this procedure to the procedure of the greedy algorithm A. Assume that the bits erased by the channel correspond to exactly those columns which in the initial step are classified as erasures. Under this assumption one can see that those columns which

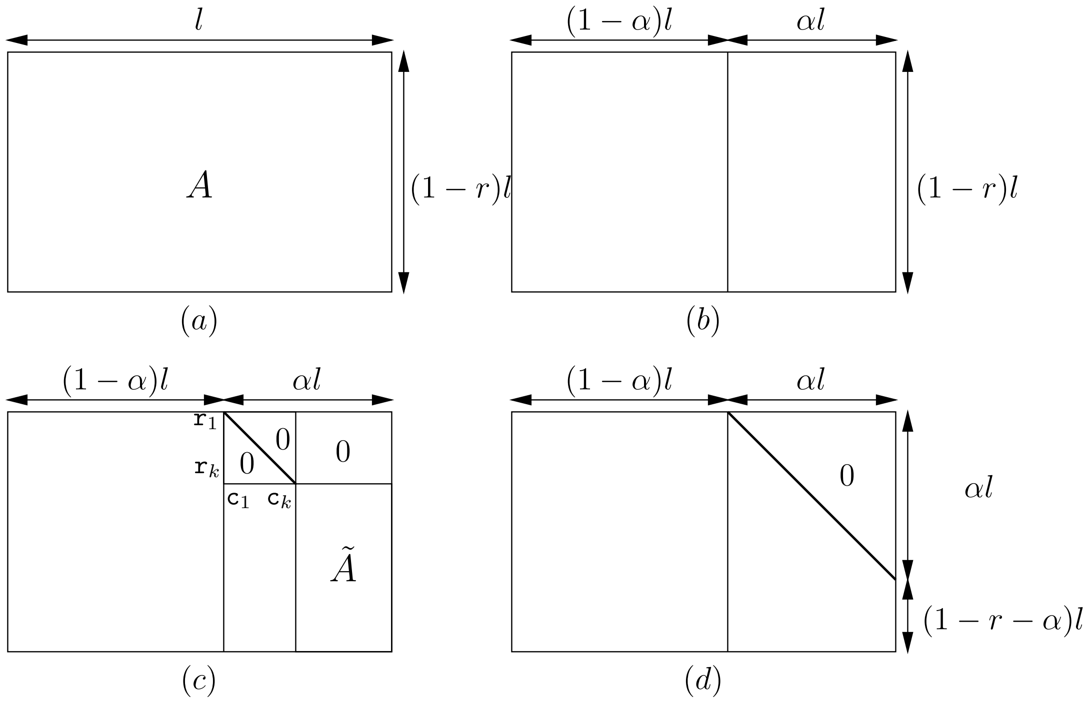

Figure 6: (a) The given matrix A. (b) After the first application of step one the $(1-\alpha)l$ known columns are reordered to form the first $(1-\alpha)l$ columns of the matrix A. (c) After the second application of step one the k new known columns and their associated rows are reordered to form a diagonal of length k. (d) If the procedure does not terminate prematurely then the diagonal is extended to have length αl and therefore the row gap is equal to $(1-r-\alpha)l$ and the column gap is equal to $(1-\alpha)l$.

are declared known in the ℓ-th round of greedy algorithm A correspond exactly to those variable nodes which are declared known in the ℓ-th round of the decoding algorithm. Hence, there is a one-to-one correspondence between these two algorithms.

As discussed in the introduction if $\alpha < \alpha^*(\mu, \nu)$ then (asymptotically in l) with high probability the decoding process will be successful. Because of the one-to-one correspondence we conclude that in this case (asymptotically in l) greedy algorithm A will extend the diagonal to (essentially) its full length αl with high probability so that the row and column gaps are as stated in the Lemma.

In the case that $r(\mu, \nu) < 0$ we can not associate an ensemble of codes to the degree distribution pair (μ, ν). Nevertheless, recursion (3) still correctly describes the expected progress of greedy algorithm A. It is also easy to see that the concentration around this expected value still occurs. It follows that the same analysis is still valid in this case. $\square$

3.1.1 Greedy Algorithm AH

By greedy algorithm AH we mean the direct application of greedy algorithm A to the extended parity check matrix $\hat{H}$ of a given LDPC code. The gap we are interested in is then simply the resulting row gap.

Corollary 1. [Performance of Greedy Algorithm AH] Let (λ, ρ) be a given degree distribution pair with $r(\lambda, \rho) \geq 0$ and choose $\alpha < \alpha^*(\lambda, \rho)$. Pick a code at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ and let $\hat{H}$ be the associated extended parity check matrix. Apply greedy algorithm A to $\hat{H}$. Then (asymptotically in n) the gap is concentrated around the value $(1 - r(\lambda, \rho) - \alpha)n$. Letting $\alpha \uparrow \alpha^*$, we see that the minimum gap achievable with greedy algorithm A is equal to $(1 - r(\lambda, \rho) - \alpha^*)n$.

Example 5. [Gap for the (3, 6)-Regular Code and Greedy Algorithm AH] From Example 2 we know that $r(x^2, x^5) = \frac{1}{2}$ and that $\alpha^*(x^2, x^5) \sim 0.429439$. It follows that the minimum expected gap size for greedy algorithm AH is equal to $g \sim 0.070561n$. $\square$

Note that greedy algorithm A establishes a link between the error correcting capability on a BEC using a message passing decoder and the encoding complexity. In simplified terms: Good codes have low encoding complexity!

3.1.2 Greedy Algorithm AHT

Rather than applying greedy algorithm A directly to the extended parity check matrix $\hat{H}$ of a LDPC code we can apply it to the *transpose* of the extended parity check matrix. In this case the gap we are interested in is equal to the resulting column gap.

Corollary 2. [Performance of Greedy Algorithm AHT] Let (λ, ρ) be a given degree distribution pair with $r(\lambda, \rho) \geq 0$ and choose $\alpha < \alpha^*(\rho, \lambda)$. Pick a code at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ and let $\hat{H}$ be the associated extended parity check matrix. Apply greedy algorithm A to $\hat{H}^T$. Recall that this is equivalent to applying greedy algorithm A to a randomly chosen extended adjacency matrix from the ensemble $\mathcal{C}^{\frac{n}{1-r(\rho,\lambda)}}(\rho, \lambda)$. Therefore, (asymptotically in n) the gap is concentrated around the value $\frac{1-\alpha}{1-r(\rho,\lambda)}n$. Letting $\alpha \uparrow \alpha^*(\rho, \lambda)$, we see that the minimum gap achievable with greedy algorithm AHT is equal to $\frac{1-\alpha^*(\rho,\lambda)}{1-r(\rho,\lambda)}n$.

Example 6. [Gap for the $(3, 6)$-Regular Code and Greedy Algorithm AHT] From Example 4 we know that $r(x^5, x^2) = -1$ and that $\alpha^*(x^5, x^2) = \frac{3^{18}}{2^{17}5^5}$. It follows that the minimum expected gap size for greedy algorithm AHT is equal to $g = \frac{1-\frac{3^{18}}{2^{17}5^5}}{2}n = \frac{22179511}{2^{18}5^5}n \sim 0.0270746n$. $\square$

Example 7. [Gap for an "Optimized" Code of Maximal Degree 12 and Greedy Algorithm AHT] Let us determine the threshold $\alpha^*(\rho, \lambda)$ for one of the "optimized" codes listed in [12]. We pick the code with $\lambda(x) := 0.251\,x + 0.309\,x^2 + 0.002\,x^3 + 0.438\,x^9$ and $\rho(x) := 0.637\,x^6 + 0.363\,x^7$. Quite surprisingly we get $\alpha^*(\rho, \lambda) = 1$! This means that for any $\epsilon > 0$ we can start the process by declaring only an $\epsilon/2$ fraction of all columns to be known and with high probability the process will continue until at most an $\epsilon/2$ fraction of all columns is left. Therefore we can achieve a gap of ϵn for any $\epsilon > 0$. We will later prove a stronger result, namely that in this case the gap is actually at most of order $\sqrt{n}$, but we will need more sophisticated techniques to prove this stronger result. $\square$

The above example shows that at least for some degree distribution pairs (μ, ν) we have $\alpha^*(\mu, \nu) = 1$. When does this happen? This is answered in the following lemma.

Lemma 2. Let (μ, ν) be a degree distribution pair. Then $\alpha^*(\mu, \nu) = 1$ if and only if for all $\alpha \in [0, 1)$

$$x > \alpha\mu(1 - \nu(1 - x)),\ \forall x \in (0, \alpha]. \tag{14}$$

Furthermore, if equation (14) holds and $\nu(0) = \mu(0) = 0$, then

$$\mu_2\nu'(1) \leq 1 \leq \mu'(1)\nu_2. \tag{15}$$

Proof. Clearly, if equation (14) holds then for any $\alpha \in [0,1)$ we have

$$x > \alpha\mu(1-\nu(1-x)),\ \forall x \in (0,\alpha].$$

By a compactness argument if follows that $\alpha_\ell(\alpha)$ as defined in (3) converges to 0 as ℓ tends to infinity. Hence, $\alpha^*(\mu,\nu) = 1$.

Assume now that $\alpha^*(\mu,\nu) = 1$. This means that for any $\alpha \in [0,1)$ we have that $\alpha_\ell(\alpha) \stackrel{\ell\to\infty}{\to} 0$. We want to show that (14) holds. Let $f(x,\alpha) := \alpha\mu(1-\nu(1-x))$ and note that for $x,\alpha \in [0,1]$, $f(x,\alpha)$ is an increasing function in both its arguments. Note that because $f(x,\alpha)$ is increasing in x it follows that a necessary condition for $\alpha_\ell(\alpha)$ to converge to zero is that $\alpha > f(\alpha,\alpha)$, i.e., that at least in the first iteration the erasure probability decreases. We will use contraposition to prove (14). Hence, assume that there exist a strictly positive x and an α, $x \leq \alpha < 1$, such that $x \leq f(x,\alpha)$. Since $\alpha > f(\alpha,\alpha)$ and since f is continuous this implies that there exists a strictly positive x and an α, $x \leq \alpha < 1$, such that $x = f(x,\alpha)$. Then $\alpha_1(\alpha) = f(\alpha,\alpha) \geq f(x,\alpha) = x$. It follows by finite induction that $\alpha_\ell(\alpha) = f(\alpha_{\ell-1},\alpha) \geq f(x,\alpha) = x$ and therefore $\alpha_\ell(\alpha)$ does not converge to zero as ℓ tends to infinity, a contradiction.

Finally, for x close to one we have $\mu(1-\nu(x)) = \mu_2\nu'(1)(1-x) + O((1-x)^2)$ whereas for x tending to zero we have $\mu(1-\nu(x)) = 1 - \mu'(1)\nu_2 x + O(x^2)$. This yields the stability conditions stated in (15). □

3.2 Greedy Algorithm B

For greedy algorithm A the elements of the initial set of known columns are chosen independently from each other. We will now show that by allowing dependency in the initial choice the resulting gap can sometimes be reduced. Of course, this dependency makes the analysis more difficult.

In order to describe and analyze greedy algorithm B we need to introduce some more notation. We call a polynomial $\omega(x) := \sum_{i\geq 1}\omega_i x^{i-1}$ with real non-negative coefficients in the range $[0,1]$ a *weight* distribution and we denote the set of all such weight distributions by $\mathcal{W}$. Let $\tau : \mathcal{D}\times\mathcal{W} \to \mathcal{D}$ be a map which maps a pair consisting of a degree distribution ν and a weight distribution ω into a new degree distribution $\tau(\nu,\omega)$. This map is defined as

$$\begin{aligned}
\tau(\nu,\omega)_1 &:= \nu_1 + \sum_{i\geq 2}\nu_i\omega_i, \\
\tau(\nu,\omega)_i &:= \nu_i(1-\omega_i),\ i \geq 2.
\end{aligned}$$

We are now ready to state greedy algorithm B.

Greedy Algorithm B:

0. [Initialization] We are given a matrix A and a weight distribution $\omega(x)$. For each row in A perform the following: if the row has weight i then select this row with probability ω_i. For each selected row of weight i declare a random subset of size $(i-1)$ of its i connected columns to be known. All remaining columns which have not been classified as known are classified as erasures. Let $\tilde{A} := A$.

1. [Stop or Extend] If $\tilde{A}$ neither contains a known column nor a row of degree one then output the present matrix. Otherwise perform a diagonal extension step.

2. [Declare Variables as Known] Any column in $\tilde{A}$ which is connected to a degree one row is declared to be known. Goto 1.

Clearly, greedy algorithm B differs from greedy algorithm A only in the choice of the initial set of columns.

Lemma 3. [Analysis of Greedy Algorithm B] Let (μ, ν) be a given degree distribution pair. Let $\omega(x)$ be a weight distribution such that $\alpha^*(\mu, \tau(\nu, \omega)) = 1$. Define $q := \sum_{i \geq 1} \nu_i \omega_i \frac{i-1}{i}$. Pick a graph at random from the ensemble $\mathcal{C}^l(\mu, \nu)$ and let $\hat{A}$ be its extended adjacency matrix. Apply greedy algorithm B to the extended adjacency matrix $\hat{A}$. Then (asymptotically in l) the row gap is concentrated around the value $\left(\frac{\sum_{i \geq 1} [1-(1-q)^i] \frac{\mu_i}{i}}{\int \mu} - r(\mu, \nu)\right) l$ and the column gap is concentrated around the value $\frac{\sum_{i \geq 1} [1-(1-q)^i] \frac{\mu_i}{i}}{\int \mu} l$.

Proof. The elements of the initial set of known columns are clearly dependent (since groups of those columns are connected to the same row) and therefore we can not apply our previous methods directly. But as we will show now there is a one-to-one correspondence between applying greedy algorithm B to the ensemble $\mathcal{C}^l(\mu, \nu)$ with a weight distribution $\omega(x)$ and applying greedy algorithm A to the transformed ensemble $\mathcal{C}^l(\mu, \tau(\nu, \omega))$.

Assume we are given the ensemble $\mathcal{C}^l(\mu, \nu)$ and a weight distribution $\omega(x)$. Assume further that we are given a fixed set of selected right nodes (rows) $\mathtt{r}_1, \ldots, \mathtt{r}_k$, and that the fraction of selected right nodes of degree i is equal to ω_i. Given a graph from $\mathcal{C}^l(\mu, \nu)$ transform it in the following way: replace each selected right node of degree i by i right nodes of degree 1. One can check that this transformation leaves the left degree distribution μ unchanged and that it transforms the right degree distribution ν to $\tau(\nu, \omega)$. Therefore the new graph is an element of the ensemble $\mathcal{C}^l(\mu, \tau(\nu, \omega))$. Further one can check that this map is reversible and, therefore, one-to-one. A closer look reveals now that applying greedy algorithm B to an extended adjacency matrix picked randomly from the ensemble $\mathcal{C}^l(\mu, \nu)$ is equivalent to applying greedy algorithm A with $\alpha = 0$ to the transformed extended adjacency matrix, i.e., the resulting residual graphs (which could be empty) will be the same. Now since $\tau(\nu, \omega)_1 > 0$ it follows that the greedy algorithm B will get started and since by assumption $\alpha^*(\mu, \tau(\nu, \omega)) = 1$ we know from the analysis of greedy algorithm A that with high probability the diagonalization process will continue until the diagonal has been extended to (essentially) its full length. In this case the resulting column gap is equal to the size of the set which was initially classified as known. To determine the size of this set we first determine the probability that a randomly chosen edge is one of those edges which connect a selected right node to one of its $(i-1)$ declared known neighbors. A quick calculation shows that this probability is equal to $q = \sum_{i \geq 1} \nu_i \omega_i \frac{i-1}{i}$. Therefore, the probability that a given left node of degree i is connected to at least one of these edges is equal to $1 - (1-q)^i$. From this the stated row and column gaps follow easily. $\square$

3.2.1 Greedy Algorithm BH

Following our previous notation, by greedy algorithm BH we mean the direct application of greedy algorithm B to the extended parity check matrix $\hat{H}$ of a given LDPC code. The gap we are interested in is then simply the resulting row gap.

Corollary 3. [Performance of Greedy Algorithm BH] Let (λ, ρ) be a given degree distribution pair with $r(\lambda, \rho) \geq 0$. Let $\omega(x)$ be a weight distribution such that $\alpha^*(\lambda, \tau(\rho, \omega)) = 1$. Define $q := \sum_{i \geq 1} \rho_i \omega_i \frac{i-1}{i}$. Pick a code at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ and let $\hat{H}$ be its extended parity check matrix. Apply greedy algorithm B to the extended parity check matrix $\hat{H}$. Then (asymptotically in n) the gap is concentrated around the value $\left(\frac{\sum_{i \geq 1} [1-(1-q)^i] \frac{\lambda_i}{i}}{\int \lambda} - r(\lambda, \rho) \right) n$. Let $q^* := \inf_{\omega(x)} \{ \sum_{i \geq 1} \rho_i \omega_i \frac{i-1}{i} : \alpha^*(\lambda, \tau(\rho, \omega)) = 1 \}$. Then we see that the minimum gap achievable with greedy algorithm BH is equal to $\left(\frac{\sum_{i \geq 1} [1-(1-q^*)^i] \frac{\lambda_i}{i}}{\int \lambda} - r(\lambda, \rho) \right) n$.

Example 8. [Gap for the (3, 6)-Regular Code and Greedy Algorithm BH] We have $\lambda(x) = x^2$ and $\rho(x) = x^5$ and since $\rho(x)$ has only one non-zero term it follows that we can parameterize $\omega(x)$ as $\omega(x) = \omega_6 x^5$. Therefore we have $\tau(\rho, \omega) = \omega_6 + (1 - \omega_6) x^5$ and since $q = \omega_6 \frac{5}{6}$ it follows that we need to find the smallest value of ω_6, call it ω_6^*, such that $\alpha^*(x^2, \omega_6 + (1-\omega_6)x^5) = 1$. From Lemma (14) we see that a necessary and sufficient condition is given by

$$1 - x > (1 - \omega_6 - (1-\omega_6)x^5)^2 = (1-\omega_6)^2(1-x^5)^2, \quad \forall x \in (0,1).$$

Equivalently we get

$$(1-\omega_6)^2 < \frac{1-x}{(1-x^5)^2}, \quad \forall x \in (0,1).$$

Differentiating shows that the right hand side takes on its minimum at the unique positive root of the polynomial $-1 - x - x^2 - x^3 + 9x^4$. If we call this root x^*, with $x^* \sim 0.739429$, then we conclude that $\omega_6^* = 1 - \sqrt{\frac{1-x_0^*}{(1-(x_0^*)^5)^2}} \sim 0.344683$. We then get $q^* = \omega_6^* \frac{5}{6} \sim 0.287236$ and therefore the gap is equal to $1 - (1-q^*)^3 - \frac{1}{2} \simeq 0.137893n$. Note that in this case the gap is larger than the corresponding gap for greedy algorithm AH. $\square$

3.2.2 Greedy Algorithm BHT

Again as for greedy algorithm A, rather than applying greedy algorithm B directly to the extended parity check matrix $\hat{H}$ of a LDPC code we can apply it to the transpose of the extended parity check matrix. In this case the gap we are interested in is equal to the resulting column gap.

Corollary 4. [Performance of Greedy Algorithm BHT] Let (λ, ρ) be a given degree distribution pair with $r(\lambda, \rho) \geq 0$. Let $\omega(x)$ be a weight distribution such that $\alpha^*(\rho, \tau(\lambda, \omega)) = 1$. Define $q := \sum_{i \geq 1} \lambda_i \omega_i \frac{i-1}{i}$. Pick a code at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$ and let $\hat{H}$ be its extended parity check matrix. Apply greedy algorithm B to $\hat{H}^T$. Recall that this is equivalent to applying greedy algorithm B to a randomly chosen extended adjacency matrix from the ensemble $\mathcal{C}^{\frac{n}{1-r(\rho,\lambda)}}(\rho, \lambda)$. Therefore (asymptotically in n) the gap is concentrated around the value $\frac{\sum_{i \geq 1}[1-(1-q)^i]\frac{\rho_i}{i}}{(1-r(\rho,\lambda))\int \rho} n$. Let $q^* := \inf_{\omega(x)}\{\sum_{i \geq 1} \lambda_i \omega_i \frac{i-1}{i} : \alpha^*(\rho, \tau(\lambda, \omega)) = 1\}$. Then we see that the minimum gap achievable with greedy algorithm BHT is equal to $\frac{\sum_{i \geq 1}[1-(1-q^*)^i]\frac{\rho_i}{i}}{(1-r(\rho,\lambda))\int \rho} n$.

Example 9. [Gap for the $(3,6)$-Regular Code and Greedy Algorithm BHT] We have $\lambda(x) = x^2$ and $\rho(x) = x^5$ and since $\lambda(x)$ has only one non-zero term we can parameterize $\omega(x)$ as $\omega(x) = \omega_3 x^2$. Therefore we have $\tau(\lambda, \omega) = \omega_3 + (1-\omega_3)x^2$ and since $q = \omega_3 \frac{2}{3}$ it follows that we need to find the smallest value of ω_3, call it ω_3^*, such that $\alpha^*(x^5, \omega_3 + (1-\omega_3)x^2) = 0$. From Lemma (14) we see that a necessary and sufficient condition is given by

$$1 - x > (1 - \omega_3 - (1-\omega_3)x^2)^5 = (1-\omega_3)^5(1-x^2)^5, \quad \forall x \in [0,1),$$

which simplifies to

$$1 > (1-\omega_3)^5(1-x^2)^4(1+x)\,.$$

By differentiating $(1-x^2)^4(1+x)$ we find that it takes its minimum at $x = 1/9$. Thus, the critical value of ω_3^* is given by $\omega_3^* = 1 - \frac{81}{80}(\frac{8}{9})^{\frac{1}{5}} = 0.01107$. We then get $q^* = \omega_3^* \frac{2}{3} \sim 0.00738$. This corresponds to a gap of $\frac{(1-(1-q^*)^6)}{2} n \simeq 0.02174n$. This is significantly better than the corresponding gap for greedy algorithm AHT.

□

3.3 Greedy Algorithm C

Let (μ, ν) be the given degree distribution pair. Recall that for greedy algorithm B we chose the weight distribution $\omega(x)$ in such a way that $\alpha^*(\mu, \tau(\nu, \omega)) = 1$. Hence, with high probability, the greedy algorithm will extend the diagonal to (essentially) its full length.

Alternatively, we can try to achieve an approximate triangulation in several smaller steps. More precisely, assume that we pick the weight distribution $\omega(x)$ in such a way that $\alpha^*(\mu, \tau(\nu, \omega)) < 1$. Then with high probability the greedy algorithm will not complete the triangulation process. Note that, conditioned on the size and on the degree distribution pair of the resulting residual graph, the edges of this residual graph are still random, i.e., if the residual graph has length $\tilde{n}$ and a degree distribution pair $(\tilde{\mu}, \tilde{\nu})$, then we can think of it as an element of $\mathcal{C}^{\tilde{n}}(\tilde{\mu}, \tilde{\nu})$. This is probably easiest seen by checking that if the destination of two edges which are contained in the residual graph are interchanged in the original graph and if the greedy algorithm B is applied to this new graph then the new residual graph will be equal to the old residual graph except for this interchange. Therefore if we achieve a triangulation by applying several small steps, then we can still use the previous tools to analyze the expected gap.

There are obviously many degrees of freedom in the choice of step sizes and the choice of weight distribution. In our present discussion we will focus on the limiting case of infinitesimal small step sizes and a constant weight distribution. Therefore, assume that we are given a fixed weight distribution $\omega(x)$ and let ϵ, $\epsilon > 0$, be a small scaling parameter for the weights such that $\alpha^*(\mu, \tau(\nu, \epsilon\omega)) < 1$. Assume that we apply greedy algorithm B to a randomly chosen element of the ensemble $\mathcal{C}^l(\mu, \nu)$ where $\mu(0) = \nu(0) = 0$. We claim that the expected degree distribution pair of the residual graph, call it $(\tilde{\mu}, \tilde{\nu})$, is given by

$$\begin{aligned}
\tilde{\mu}_i &\leftarrow \mu_i + \mu_i \frac{(\sum_{i\geq 2} \nu_i\omega_i)(\sum_{j\geq 2} \mu_j j - i)}{1-\mu'(1)\nu_2}\epsilon + O(\epsilon^2) \\
\tilde{\nu}_i &\leftarrow \nu_i + \nu_i(\sum_{j\geq 2} \omega_j\nu_j - \omega_i)\epsilon + (i\nu_{i+1} - (i-1-\nu_2)\nu_i)\frac{\mu'(1)\sum_{j\geq 2}\nu_j\omega_j}{1-\mu'(1)\nu_2}\epsilon + O(\epsilon^2)
\end{aligned}$$

To see this first recall from the analysis of greedy algorithm B that the degree distribution pair of the equivalent transformed graph is equal to $(\mu, \tau(\nu, \epsilon\omega))$. Since by assumption $\alpha^*(\mu, \tau(\nu, \epsilon\omega)) < 1$, the recursion given in (3) (with $\alpha = 1$) will have a fixed point, i.e., there exists a real number x, $x < 1$, such that

$$\mu(1 - \tau(\nu, \epsilon\omega)(1-x)) = x.$$

To determine this fixed point note that if we expand the above in $(1-x)$ around $(1-x) = 0$ we obtain

$$\mu(1-\tau(\nu,\epsilon\omega)(1-x)) = 1 - \mu'(1)[\epsilon\sum_{i\geq 2}\nu_i\omega_i + \nu_2(1-\epsilon\omega_2)(1-x)] + O((1-x)^2).$$

Therefore letting Δx denote $1-x$, the fixed point equation is

$$\Delta x = \mu'(1)[\epsilon\sum_{i\geq 2}\nu_i\omega_i + \nu_2\Delta x] + O((\Delta x)^2) + O(\epsilon\Delta x).$$

It follows that

$$\Delta x = \frac{\mu'(1)\sum_{i\geq 2}\nu_i\omega_i}{1-\mu'(1)\nu_2}\epsilon + O(\epsilon^2).$$

In the language of message passing algorithms, $1 - \Delta x$ is the expected fraction of erasure messages passed from left to right at the time the algorithm stops. The fraction of erasure messages which are passed at that time from right to left is then

$$1 - \tau(\nu, \epsilon\omega)(\Delta x) = 1 - \frac{\sum_{i\geq 2}\nu_i\omega_i}{1-\mu'(1)\nu_2}\epsilon + O(\epsilon^2). \tag{16}$$

We start by determining the residual degree distribution of left nodes. Note that a left node will not appear in the residual graph iff at least one of its incoming messages is not an erasure – otherwise it stays and retains its degree. Using (16) we see that a node of degree i has a probability of

$$\frac{\sum_{j\geq 2}\nu_j\omega_j}{1-\mu'(1)\nu_2}i\epsilon + O(\epsilon^2)$$

of being expurgated. Since in the original graph the number of left degree i nodes is proportional to $\frac{\mu_i}{i}$ it follows that in the residual graph the number of left degree i nodes is proportional to

$$\frac{\mu_i}{i}[1 - \frac{\sum_{j\geq 2}\nu_j\omega_j}{1-\mu'(1)\nu_2} i\epsilon + O(\epsilon^2)].$$

From an edge perspective the degree i fraction of the residual graph is therefore proportional to

$$\mu_i[1 - \frac{\sum_{j\geq 2}\nu_j\omega_j}{1-\mu'(1)\nu_2} i\epsilon + O(\epsilon^2)].$$

After normalization we find that the left degree distribution of the residual graph, call it $\tilde{\mu}$, is given by

$$\tilde{\mu}_i \leftarrow \mu_i[1 + \frac{(\sum_{j\geq 2}\nu_j\omega_j)(\sum_{j\geq 2}\mu_j j - i)}{1-\mu'(1)\nu_2}\epsilon + O(\epsilon^2)].$$

We next determine the right degree distribution of the residual graph. Recall that the equivalent transformed graph has a right degree distribution of $\tau(\nu, \epsilon\omega)$. We are only interested in nodes of degree at least two. Hence we have

$$\tau(\nu, \epsilon\omega)_i \quad \propto \quad \nu_i(1 - \epsilon\omega_i),\ i \geq 2.$$

From a node perspective these fractions are proportional to

$$\frac{\nu_i}{i}(1 - \epsilon\omega_i),\ i \geq 2.$$

Define the *erasure degree* of a right node to be equal to the number of incoming edges which carry erasure messages. To first order in ϵ a node of erasure degree i can stem either from a node of regular degree i all of whose incoming messages are erasures or it can stem from a node of regular degree $(i+1)$ which has one non-erasure message. Hence, at the fixed point the fraction of right nodes with an erasure degree of i is proportional to

$$\frac{\nu_i}{i}(1-\epsilon\omega_i)(1-\Delta x)^i + \frac{\nu_{i+1}}{i+1}(1-\epsilon\omega_{i+1})(i+1)\Delta x(1-\Delta x)^i + O(\epsilon^2).$$

Converting back to an edge perspective we see that these fractions are proportional to

$$\begin{aligned} & \nu_i(1-\epsilon\omega_i)(1-\Delta x)^i + \nu_{i+1} i(1-\epsilon\omega_{i+1})\Delta x(1-\Delta x)^i + O(\epsilon^2) \\ = \quad & \nu_i(1-(\epsilon\omega_i + i\Delta x)) + \nu_{i+1} i\Delta x + O(\epsilon^2)\,. \end{aligned}$$

Summing the above over $i \geq 2$ we obtain $1 - \epsilon \sum_{i \geq 2} \nu_i \omega_i - (1 + \nu_2)\Delta x$. Noting that $\Delta x = O(\epsilon)$ and normalizing we see that the residual right degree distribution, call it $\tilde{\nu}$, is given by

$$\begin{aligned}
\tilde{\nu}_i &\leftarrow \nu_i(1 - \epsilon(\omega_i - \sum_{j \geq 2} \nu_j \omega_j) - (i - 1 - \nu_2)\Delta x)) + \nu_{i+1} i \Delta x + O(\epsilon^2) \\
&= \nu_i + \nu_i(\sum_{j \geq 2} \omega_j \nu_j - \omega_i)\epsilon + (i\nu_{i+1} - (i - 1 - \nu_2)\nu_i)\frac{\mu'(1) \sum_{j \geq 2} \nu_j \omega_j}{1 - \mu'(1)\nu_2}\epsilon + O(\epsilon^2)
\end{aligned}$$

We are ultimately interested in the resulting row and column gaps. Since one can easily be determined from the other we will only write down an expression for the row gap. If we take the expression for the row gap, call it g_r from greedy algorithm B and keep only the terms which are linear in ϵ then we see that the row gap increased according to

$$g_r \leftarrow g_r + \frac{\sum_{j \geq 2} \nu_j \omega_j \frac{j-1}{j}}{\sum_{j \geq 2} \frac{\mu_j}{j}} l \epsilon + O(\epsilon^2)$$

The length l of the code itself evolves as

$$l \leftarrow l[1 - \frac{\sum_{j \geq 2} \nu_j \omega_j}{(1 - \mu'(1)\nu_2) \sum_{j \geq 2} \frac{\mu_j}{j}}\epsilon] + O(\epsilon^2).$$

Collecting all results we see that as a function of the independent variable ϵ all quantities evolve according to the system of differential equations:

$$\begin{aligned}
\frac{\partial \mu_i}{\partial \epsilon} &= \mu_i \frac{(\sum_{i \geq 2} \nu_i \omega_i)(\sum_{j \geq 2} \mu_j j - i)}{1 - \mu'(1)\nu_2} \\
\frac{\partial \nu_i}{\partial \epsilon} &= \nu_i(\sum_{j \geq 2} \omega_j \nu_j - \omega_i) + (i\nu_{i+1} - (i - 1 - \nu_2)\nu_i)\frac{\mu'(1) \sum_{j \geq 2} \nu_j \omega_j}{1 - \mu'(1)\nu_2} \\
\frac{\partial g_r}{\partial \epsilon} &= \frac{\sum_{j \geq 2} \nu_j \omega_j \frac{j-1}{j}}{\sum_{j \geq 2} \frac{\mu_j}{j}} l \\
\frac{\partial l}{\partial \epsilon} &= -\frac{\sum_{j \geq 2} \nu_j \omega_j}{(1 - \mu'(1)\nu_2) \sum_{j \geq 2} \frac{\mu_j}{j}}
\end{aligned}$$

with the value of the initial quantities equal to μ, ν, 0, and l, respectively.

As before we can apply greedy algorithm C directly to the extended parity check matrix chosen randomly from an ensemble $\mathcal{C}^n(\lambda, \rho)$ in which case we are interested in the resulting column gap or we can apply it to the transpose of the extended parity check matrix in which case we are interested in the column gap. We call these algorithms CH and CHT, respectively.

Example 10. [Gap for (3,6)-Regular Code and Greedy Algorithm CHT.] We choose $\omega_2 = 1$ and let $\omega_3 = \eta$ where η is some very small quantity. Solving the system of differential equations reveals that the resulting gap is equal to $0.017n$. We see that this is the smallest expected gap for all the presented algorithms. $\square$

3.4 A Practical Greedy Algorithm

In practice one implements a serial version of greedy algorithm CHT. At each stage, if the residual graph has a degree one variable node then diagonal extension is applied. If no such degree one variable node exists then one selects a variable node of lowest possible degree, i say, from the residual graph, and declares $i-1$ (assuming no multiple edges) of its neighbors to be known. The residual graph now has at least one degree one node and diagonal extension is applied.

There are many practical concerns. For example, variable nodes which are used in the diagonal extension step correspond to non-systematic variables. Typically, degree two nodes have the highest bit error rates. Thus, it is preferable to use as many low degree variables in the diagonalization step as possible. E.g., if the subgraph induced by only the degree two variables has no loops then all degree two variables can be made non-systematic using the above algorithm.

4 Codes with Linear Encoding Complexity

We saw in the previous section that degree distributions giving rise to codes that allow transmission close to capacity will have gaps that are smaller than an arbitrarily small linear fraction of the length of the code. To prove that these codes have linear encoding complexity more work is needed, namely one has to show that the gap g satisfies $g \leq O(\sqrt{n})$ with high probability for large enough n. More precisely, we will prove

Theorem 1. [Codes with Linear Encoding Complexity] Let (λ, ρ) be a degree distribution pair satisfying $\alpha^*(\rho, \lambda) = 1$, with minimum right degree $\underline{\mathtt{d}}_{\mathtt{c}} > 2$, and satisfying the *strict* inequality $\lambda'(0)\rho'(1) > 1$. Let $\mathtt{G}$ be chosen at random from the ensemble $\mathcal{C}^n(\lambda, \rho)$. Then $\mathtt{G}$ is encodable in linear time with probability at least $1 - bc^{\sqrt{n}}$ for some positive constants b and c, where $c < 1$.

Discussion. We note that all optimized degree distribution pairs listed in [12] fulfill the conditions of Theorem 1. Furthermore, in experiments when applying the practical greedy algorithm to graphs based on these degree distribution pairs the resulting gap is typically in the range of one to three!

This is true even for very large lengths like one million. By correctly choosing the first degree-two variable, the gap can nearly always be lowered to one. The primary reason for these very small gaps is the large number of degree two variable nodes in these degree distributions. The number of degree two variable nodes is sufficiently large so that, with very high probability, the subgraph induced by these nodes has a large (linear size) connected component. Once a single check node belonging to this component is declared known then the remainder of the component will diagonalize in the next diagonal extension step. The diagonalization process then typically completes without further increasing the gap. □

Proof. In order to show that under the stated conditions elements of the ensembles $\mathcal{C}^n(\lambda, \rho)$ are linear time encodable (with high probability) it suffices to show that their corresponding $\hat{H}^T$ can be brought into approximate lower-triangular form with a gap of no more than $O(\sqrt{n})$ (with high probability). Note that we are working on the *transpose* of the parity check matrix. Although one can prove that such an approximate triangulation is achieved by the *practical greedy algorithm* it will be more convenient to consider a slightly different greedy algorithm.[4] The algorithm we consider has three phases which we will have to investigate separately: *start-up*, *main triangulation*, *clean-up*. In the start-up phase we will declare at most $O(\sqrt{n})$ of the check nodes to be known. Each time we declare one check node to be known we apply the diagonal extension step repeatedly until either there are no degree one variable nodes left or until (we hope) the number of degree one variable nodes has grown to a linear sized fraction. Assuming the latter, we then enter the *main triangulation* process. With exponential probability the process will continue until we are left with at most a small linear fraction of nodes. Now we enter the *clean-up* phase. Here, we will show that with high probability at most $O(\sqrt{n})$ check nodes will be left when the algorithm terminates. So overall, with high probability the gap will be no more than $O(\sqrt{n})$, which will prove the claim. We will now discuss these three phases in detail.

Recall that our aim is to bring a given $\hat{H}^T$, where $\hat{H}$ is a random element from $\mathcal{C}^n(\lambda, \rho)$, into approximate lower-triangular form with gap at most $O(\sqrt{n})$ by applying a greedy algorithm.

Start-Up: Let $\mathtt{v}$ be a randomly chosen degree two variable node and let $\mathtt{c}_1$ and $\mathtt{c}_2$ be its connected check nodes. Declare $\mathtt{c}_1$ to be *known*. Now perform the diagonal extension step. After this step the columns which correspond to $\mathtt{c}_1$ and $\mathtt{c}_2$ will form the first two columns of the matrix (assuming

[4]In appendix C we define the notion of a 'stopping set'. Stopping sets determine the termination points of diagonal extension steps regardless of the implementation of the diagonal extension. Thus, the particular three phase formulation used here is only for convenience of presentation.

$\mathtt{v}$ does not have a double edge) and the row corresponding to $\mathtt{v}$ will form the first row of the matrix. Consider the residual matrix (with the first two columns and the first row deleted) and the corresponding residual graph. If this residual matrix contains a degree one row then we can apply another diagonal extension step and so on. It will simplify our description if we perform the diagonal extension step to *one degree one variable node at a time*, instead of to all degree one variable nodes in parallel. More precisely, we start out with one degree two variable node which we convert into a degree one variable node by declaring one of its neighbors to be known. Then at any stage of the procedure choose one of the degree one variable nodes (assuming that at least one such node exists) and perform the diagonal extension step only on this variable.

Let Y_{t} denote the number of degree one variable nodes after the t-th such step, where we have $Y_0 = 1$. If by X_{t} we denote the number of *additional* degree one variable nodes which are generated in the t-th step then we get

$$Y_{\mathrm{t}} = Y_{\mathrm{t}-1} + X_{\mathrm{t}} - 1, \tag{17}$$

where the minus one term stems from the fact that one degree one variable node is used up during the diagonal extension step. Equation (17) is an instance of a *branching process*. Note that the process continues until $Y_{\mathrm{t}} = 0$, i.e., until there are no more degree one variable nodes available for the diagonal extension step. We would like the process to continue until Y_{t} has reached "linear size", i.e., until Y_{t} is a small fixed fraction of the number of variable nodes.

Assume that we have performed at most $\epsilon\, n$ steps. Let $(\lambda^{\mathrm{res}}, \rho^{\mathrm{res}})$ denote the *residual* degree distribution pair. If ϵ is small it is intuitively clear that $(\lambda^{\mathrm{res}}, \rho^{\mathrm{res}})$ is "close" to (λ, ρ). Indeed, in Lemma 4 in Appendix A it is shown that, given a degree distribution pair (λ, ρ) such that $\lambda_2 \rho'(1) > 1$, then there exists an $\epsilon > 0$ and an $\gamma > 1$ such that $\lambda_2^{\mathrm{res}} {\rho^{\mathrm{res}}}'(1) \geq \gamma > 1$ regardless which check nodes have been removed, as long as their total number is no more than $\epsilon\, n$.

So, assume that we have performed at most $\epsilon\, n$ steps. What is the expected value of X_{t}? Consider an edge $\mathtt{e}$ emanating from a degree one variable node. With probability ρ_j^{res} it is connected to a degree j check node, call this node $\mathtt{c}$. This check node has $(j-1)$ other edges, each of which has probability λ_2^{res} of being connected to a degree two node. Therefore, if $\mathtt{c}$ has degree j then the expected number of *new* degree one nodes that will be generated is equal to $(j-1)\lambda_2^{\mathrm{res}}$. Averaging over all degrees we get that X_{t} has expected value $\lambda_2^{\mathrm{res}} {\rho^{\mathrm{res}}}'(1) \geq \gamma > 1$. In other words we have $\mathbb{E}[X_{\mathrm{t}} | X_1, \cdots, \cdots, X_{\mathrm{t}-1}] \geq \gamma$ for $1 \leq \mathrm{t} \leq \epsilon\, n$. Furthermore, X_{t} is upper bounded by the maximum right degree $\overline{\mathrm{d}}_{\mathtt{c}}$.

Let us define $\mathrm{T} := \min_t\{Y_t = 0\}$ to be the stopping time of Y_t. We will say that the branching process *stops prematurely* if $\mathrm{T} \leq \sqrt{n}$ and we will say that it is *successful* if $\mathrm{T} \geq \frac{\epsilon}{2}n$ and $Y_{\frac{\epsilon}{2}n} \geq \eta\frac{\epsilon}{2}n$, where η can be chosen freely in the range $0 < \eta < 1-\gamma$. Assume now that we employ the following strategy: Start a process, by choosing a degree two variable node and declaring one of its neighbors to be known. If this process stops prematurely then start another process if the number of prematurely stopped processes so far is less than $\frac{\epsilon}{2}\sqrt{n}$ or declare a failure otherwise. If the current process has not stopped prematurely then declare a success if $\mathrm{T} \geq \frac{\epsilon}{2}n$ and $Y_{\frac{\epsilon}{2}n} \geq \eta\frac{\epsilon}{2}n$ and stop the process at that time, and declare a failure otherwise. Note that the total number of steps taken for this strategy is at most $\sqrt{n}\frac{\epsilon}{2}\sqrt{n} + \frac{\epsilon}{2}n = \epsilon n$. Although the branching process which we consider always stops at a finite time and although we will only be interested in the process for at most ϵn steps it is convenient to think of an infinite process $Y_0, Y_1, \cdots$ with the property that $1 < \gamma \leq \mathbb{E}[X_{\mathrm{t}}|X_1, \cdots, X_{\mathrm{t}-1}] < \overline{\mathrm{d}}_{\mathsf{c}}$, $l \geq 1$. This will allow us to write statements like $\Pr\{\mathrm{T} < \sqrt{n}\} \leq \Pr\{\mathrm{T} < \infty\}$.

The probability of failure can be easily bounded in the following way. From (iv) Lemma 5 in Appendix B the probability that a process has stopping time less than $\sqrt{n}$, i.e., $\Pr\{\mathrm{T} < \sqrt{n}\}$, can be upper bounded by

$$\Pr\{\mathrm{T} < \sqrt{n}\} \leq \Pr\{\mathrm{T} < \infty\} \leq \mathrm{p}(\gamma, \overline{\mathrm{d}}_{\mathsf{c}}) < 1.$$

Therefore the probability that $\frac{\epsilon}{2}\sqrt{n}$ processes have stopping time less than $\sqrt{n}$ has an upper bound of the form $c_1^{\sqrt{n}}$, $c_1 < 1$. Using (i), (ii) and (iii) of Lemma 5, the probability that a process failed assuming that it did not stop prematurely can be upper bounded as follows

$$\begin{aligned}
& \Pr\{\{\mathrm{T} \leq \frac{\epsilon}{2}n\} \cup \{Y_{\frac{\epsilon}{2}n} \leq \eta\frac{\epsilon}{2}n\}|\, \mathrm{T} \geq \sqrt{n}\} \\
\leq\ & \Pr\{\mathrm{T} \leq \frac{\epsilon}{2}n|\, \mathrm{T} \geq \sqrt{n}\} + \Pr\{Y_{\frac{\epsilon}{2}n} \leq \eta\frac{\epsilon}{2}n|\, \mathrm{T} \geq \sqrt{n}\} \\
\leq\ & \Pr\{\mathrm{T} < \infty|\, \mathrm{T} \geq \sqrt{n}\} + \frac{\Pr\{Y_{\frac{\epsilon}{2}n} \leq \eta\frac{\epsilon}{2}n\}}{\Pr\{\mathrm{T} \geq \sqrt{n}\}} \\
\leq\ & \Pr\{\mathrm{T} < \infty|\, \mathrm{T} \geq \sqrt{n}\} + \frac{\Pr\{Y_{\frac{\epsilon}{2}n} \leq \eta\frac{\epsilon}{2}n\}}{\Pr\{\mathrm{T} = \infty\}} \\
\leq\ & b_1\xi^{\sqrt{n}} + b_2 c_2^n \\
\leq\ & b_3\xi^{\sqrt{n}},
\end{aligned}$$

for some positive constants b_1, b_2, b_3 and $c_2 < 1$ and a constant $\xi < 1$ defined in Appendix B. Combining these two results, we conclude that the probability of failure is upper bounded by $b_4 c_3^{\sqrt{n}}$ for some constant b_4 and $c_3 < 1$.

Main Upper Triangulation: With high probability we will have succeeded in the start-up phase. Consider now the output of this start-up phase assuming it was successful. From Lemma 4 in Appendix A we know that the residual degree distribution pair fulfills $\underline{\mathtt{d}}_{\mathtt{c}}^{\mathrm{res}} > 2$ and $\alpha^*(\rho^{\mathrm{res}}, \lambda^{\mathrm{res}}) = 1$. Furthermore, it is easy to see that, conditioned on $(\lambda^{\mathrm{res}}, \rho^{\mathrm{res}})$ and the length $\mathrm{n}^{\mathrm{res}}$, the resulting residual graph can be thought of as an element of $\mathcal{C}^{\mathrm{n}^{\mathrm{res}}}(\lambda^{\mathrm{res}}, \rho^{\mathrm{res}})$. This is most easily seen as follows: Given the original graph the residual graph is the result of removing a certain set of edges, where the choice of these edges is the result of certain random experiments. Consider another element of $\mathcal{C}^n(\lambda, \rho)$ which agrees with the original graph in those edges but is arbitrary otherwise. Assume now that we run the start-up phase on this new graph *with the same random choices*. It is easy to see that the sequence of degree distribution pairs will be the same and that at each step the probability for the given choice of node which gets chosen is identical. So the resulting residual graphs will have identical degree distribution pairs, identical length and the same probability of being generated. Further, each element of $\mathcal{C}^{\mathrm{n}^{\mathrm{res}}}(\lambda^{\mathrm{res}}, \rho^{\mathrm{res}})$ is reachable and by the above discussion they have equal probability of being generated, which proves the claim. Since $\lambda_1^{\mathrm{res}} > 0$ it follows that we can now simply use greedy algorithm AHT to continue the lower-triangulation process. From the analysis of greedy algorithm AHT we know that with exponential probability the process will not stop until at most a small linear fraction σ of check nodes is left, where this fraction σ can be made as small as desired.

Clean-Up Operation: So far we have increased the gap to at most $\frac{\epsilon}{2}\sqrt{n}$ and with high probability we have accomplished a partial lower-triangulation with at most a small fraction σ of check nodes left. In Lemma 6 in Appendix C it is now shown that with probability $1 - \frac{1}{2}^{\sqrt{n}}$ actually fewer than $\sqrt{n}$ check nodes will be left.

Combining all these statements we see that the probability that the gap exceeds $2\sqrt{n}$ is at most $bc^{\sqrt{n}}$, where $c < 1$. $\square$

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, Massachusetts: M.I.T. Press, 1963.

[2] V. Zyablov and M. Pinsker, "Estimation of the error-correction complexity of Gallager low-density codes," *Problemy Peredachi Informatsii*, vol. 11, pp. 23–26, Jan. 1975.

[3] G. A. Margulis, "Explicit construction of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.

[4] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, Sept. 1981.

[5] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, August 1996.

[6] N. Wiberg, *Codes and Decoding on General Graphs.* PhD thesis, Linköping University, S-581 83, Linköping, Sweden, 1996.

[7] N. Sourlas, "Spin-glass models as error-correcting codes," *Nature*, vol. 339, pp. 693–695, June 1989.

[8] I. Kantor and D. Saad, "Error-correcting codes that nearly saturate Shannon's bound," *Phys.Rev.Lett.*, vol. 83, pp. 2660–2663, 1999.

[9] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the* 29*th annual ACM Symposium on Theory of Computing*, pp. 150–159, 1997.

[10] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 249–258, 1998.

[11] T. Richardson and R. Urbanke, "The capacity of low-density parity check codes under message-passing decoding." submitted IEEE IT.

[12] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of provably good low-density parity check codes." submitted IEEE IT.

[13] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. on Information Theory*, vol. 42, Nov. 1996.

[14] D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," in *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing, Sept. 1998*, 1998.

[15] D. Spielman, "Linear-time encodeable and decodable error-correcting codes," *IEEE Trans. on Information Theory*, vol. 42, Nov. 1996.

[16] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 364–373, 1998.

[17] L. Bazzi, T. Richardson, and R. Urbanke, "Exact thresholds and optimal codes for the binary symmetric channel and Gallager's decoding algorithm A." submitted IEEE IT.

[18] N. Alon, J. Spencer, and P. Erdös, *The Probabilistic Method.* New York: John Wiley & Sons, Inc., 1992.

[19] A. Schrijver, *Theory of linear and integer programming.* Wiley, 1986.

A Residual Degree Distribution Pairs

Lemma 4. Let (λ, ρ) be a degree distribution pair satisfying $\underline{\mathrm{d}}_{\mathsf{c}} > 2$, the strict inequality $\lambda_2\rho'(1) > 1$ and $\alpha^*(\rho, \lambda) = 1$. Let B_ϵ be the set of all residual degree distribution pairs obtainable from (λ, ρ) by removing at most an ϵ fraction of check nodes from a graph with degree distribution pair (λ, ρ). Then, for ϵ sufficiently small, there exists a $\gamma > 1$ such that any $(\lambda^{\text{res}}, \rho^{\text{res}}) \in B_\epsilon$ will satisfy $\underline{\mathrm{d}}_{\mathsf{c}}^{\text{res}} > 2$ and the strict inequality $\lambda_2^{\text{res}}\rho^{\text{res}\prime}(1) > \gamma$. If, moreover, for some $\alpha > 0$ we have $\lambda_1^{\text{res}} \geq \alpha\epsilon$ then $\alpha^*(\rho^{\text{res}}, \lambda^{\text{res}}) = 1$.

Proof. The conclusion $\underline{\mathrm{d}}_{\mathsf{c}}^{\text{res}} > 2$ is immediate since $\underline{\mathrm{d}}_{\mathsf{c}} > 2$ and since we either remove a check node completely or leave its degree unchanged. By continuity and since, for some γ, $\lambda_2\rho'(1) > \gamma > 1$ it is also clear that $\lambda_2^{\text{res}}{\rho^{\text{res}}}'(1) > \gamma$ if ϵ is sufficiently small.

It remains to show that $\alpha^*(\rho^{\text{res}}, \lambda^{\text{res}}) = 1$ if $\lambda_1^{\text{res}} \geq \eta\epsilon$ for some $\eta > 0$. Let $\Delta := \max\{\overline{\mathrm{d}}_{\mathsf{v}}, \overline{\mathrm{d}}_{\mathsf{c}}\}$. Let δ be a positive number such that $\lambda_2\rho'(1) > 1 + \delta$. It follows by continuity that for ϵ small enough we have $\lambda_2^{\text{res}}{\rho^{\text{res}}}'(1 - \lambda_1^{\text{res}}) > 1 + \delta/2$. Define $\gamma(x) := \rho^{\text{res}}(1 - \lambda^{\text{res}}(x))$ and note that $\sum |\gamma_i| \leq 2^\Delta$. Since ${\rho^{\text{res}}}'(1) \geq 2$ we have $\rho^{\text{res}}(1 - \lambda_1^{\text{res}}) \leq 1 - \eta\epsilon$ for ϵ sufficiently small. Hence, for ϵ sufficiently small, we have

$$
\begin{aligned}
\rho^{\text{res}}(1 - \lambda^{\text{res}}(x)) &\leq \rho^{\text{res}}(1 - \lambda_1^{\text{res}}) - {\rho^{\text{res}}}'(1 - \lambda_1^{\text{res}})\lambda_2^{\text{res}} x + 2^\Delta x^2 \\
&\leq 1 - \eta\epsilon - x(1 + \delta/2 - 2^\Delta x) \\
&\leq 1 - x - \eta\epsilon, \quad \forall x \in [0, \delta/2^{\Delta+1}).
\end{aligned}
$$

In a similar manner

$$\begin{aligned} \rho^{\text{res}}(1-\lambda^{\text{res}}(x)) &\leq \rho_2^{\text{res}}\lambda^{\text{res}\prime}(1)(1-x)+2^{\Delta}(1-x)^2 \\ &\leq 2^{\Delta}(1-x)^2 \\ &< 1-x, \quad \forall x \in (1-2^{-\Delta},1). \end{aligned}$$

In the compact range $[\delta/2^{\Delta+1}, 1-2^{-(\Delta+1)}]$, $\rho^{\text{res}}(1-\lambda^{\text{res}}(x))$ is a continuous function in the perturbation and since the degree distribution pair (λ,ρ) fulfills the strict inequality (14) in this range it follows that there exists an ϵ_2 such that it $\epsilon \leq \epsilon_2$ then $\rho^{\text{res}}(1-\lambda(x)) < 1-x-\epsilon$ in this range. Let us further assume $\epsilon < 2^{-(\Delta+2)}$. Then it follows that on the interval $[1-2^{-(\Delta+1)}, 1-2\epsilon]$ we have $2^{\Delta}(1-x)^2-(1-x)+\epsilon < 0$ and hence $\rho^{\text{res}}(1-\lambda(x)) < 1-x-\epsilon$. This shows that $\alpha^*(\rho^{\text{res}},\lambda^{\text{res}}) = 1$. □

B Branching Processes

Let $X_1, X_2, \cdots$ be a sequence of i.i.d. random variables. Define the sequence $Y_0, Y_1, \cdots$ by the recursion $Y_0 := 1$ and $Y_{\mathrm{t}} := Y_{\mathrm{t}-1} + X_{\mathrm{t}} - 1$ and let T be the least time t such that $Y_{\mathrm{t}} = 0$. If no such time t exists then we define $\mathrm{T} = \infty$. The process $Y_0, Y_1, \cdots$ is usually referred to as a *branching process* and originated in the context of the study of population growths, [18]. It is well known that if $\mathbb{E}[X_{\mathrm{t}}] < 1$ then $\Pr\{\mathrm{T} = \infty\} = 0$ but that if $\mathbb{E}[X_{\mathrm{t}}] > 1$ then $\Pr\{\mathrm{T} = \infty\} > 0$. We will now show that under suitable conditions the same conclusions hold even if we allow (limited) dependency within the process $X_1, X_2, \cdots$.

Lemma 5. [Branching Processes] Let $X_1, X_2, \cdots$, be a sequence of random variables taking values in $\{0, \cdots, d\}$ such that $1 < \gamma \leq \mathbb{E}[X_{\mathrm{t}} | X_1, \cdots, X_{\mathrm{t}-1}] \leq d$ for all $\mathrm{t} \geq 1$. Define the *branching process* $Y_0, Y_1, \cdots$, by $Y_0 := 1$ and $Y_{\mathrm{t}} := Y_{\mathrm{t}-1} + X_{\mathrm{t}} - 1$, $\mathrm{t} \geq 1$. Let the *stopping time* T be defined by $\mathrm{T} := \min_{\mathrm{t}\geq 1}\{\mathrm{t} : Y_{\mathrm{t}} = 0\}$, where $\mathrm{T} := \infty$ if no such t exists.

(i) For any $\alpha \in (0,1)$

$$\Pr\{Y_{\mathrm{t}} \leq (\alpha\gamma - 1)\,\mathrm{t}\} \leq e^{-\mathrm{t}\,\frac{\gamma}{d}(1-\alpha(1-\ln\alpha))}.$$

(ii) Define $\xi := \xi(\gamma, d) := e^{-\frac{1}{d}(\gamma-1-\ln\gamma)} < 1$. Then

$$\Pr\{\mathrm{T} < \infty | \, \mathrm{T} \geq \mathrm{t}\} \leq \frac{\xi^t}{1-\xi}.$$

(iii) For any $\mathrm{t} \geq 0$

$$\Pr\{\mathrm{T} > \mathrm{t}\} > \left(\frac{\gamma}{d}\right)^{\mathrm{t}}.$$

(iv) Define $\mathrm{p} := \mathrm{p}(\gamma, d) := \min_{\mathrm{t}\geq 1} 1 - \frac{1-\xi-\xi^{\mathrm{t}}}{1-\xi}\left(\frac{\gamma}{d}\right)^{\mathrm{t}} < 1$. Then

$$\Pr\{\mathrm{T} < \infty\} \leq \mathrm{p}\,.$$

Proof. We start with a proof of the tail inequality. For any $s > 0$ we have

$$\begin{aligned}
\Pr\{Y_t \leq (\alpha\gamma - 1)\,\mathrm{t}\} &\leq \Pr\{\sum_{i=1}^{\mathrm{t}} X_i \leq \alpha\gamma m\} \\
&= \Pr\{-\sum_{i=1}^{\mathrm{t}} X_i \geq -\alpha\gamma\,\mathrm{t}\} \\
&= \Pr\{e^{-s\sum_{i=1}^{\mathrm{t}} X_i} \geq e^{-s\alpha\gamma\,\mathrm{t}}\} \\
&\leq e^{s\alpha\gamma\,\mathrm{t}}\mathbb{E}[e^{-s\sum_{i=1}^{\mathrm{t}} X_i}],
\end{aligned}$$

where the last step follows from the well known Markov inequality. We proceed by bounding $\mathbb{E}[e^{-s\sum_{i=1}^{\mathrm{t}} X_i}]$.

Recall the following basic fact from the theory of linear programming, [19]. A *primal* linear program in *normal form*, $\min\{cu | u \geq 0; Au \geq b\}$, has the associated *dual* linear program $\max\{vb | v \geq 0; vA \leq c\}$. Further, if u^* and v^* are feasible solutions for the primal and dual linear program such that their *values* are equal, i.e., $cu^* = v^*b$, then u^* and v^* are *optimal* solutions.

Now note that

$$\begin{aligned}
&\max_{u_k}\{\mathbb{E}[e^{-sX_i}|X_1, \cdots, X_{i-1}] | \mathbb{E}[X_i|X_1, \cdots, X_{i-1}] \geq \gamma; X_i \in \{0, \cdots, d\}\} \\
\leq\ &\max_{u_k}\{\sum_{k=0}^{d} e^{-sk} u_k | u_k \geq 0; \sum_{k=0}^{d} k u_k \geq \gamma; \sum_{k=0}^{d} u_k = 1\} \\
=\ &-\min_{u_k}\{-\sum_{k=0}^{d} e^{-sk} u_k | u_k \geq 0; \sum_{k=1}^{d} k u_k \geq \gamma; -\sum_{k=0}^{d} u_k \geq -1\}.
\end{aligned}$$

The last step warrants some remarks. First we rewrote the maximization as a minimization to bring the linear program into normal form. Second, a simple scaling argument shows that one can replace the equality condition $\sum_{k=0}^{d} u_k = 1$ with the inequality $\sum_{k=0}^{d} u_k \leq 1$ without changing the value of the linear program. The linear program in the last line is our primal problem. It is easy to check that this primal problem has the feasible solution

$$u_0 = 1 - \frac{\gamma}{d}; u_k = 0, 1 \leq k \leq d-1; u_d = \frac{\gamma}{d},$$

with value

$$-(1-\frac{\gamma}{d}(1-e^{-sd})). \tag{18}$$

To see that this solution is optimal consider the associated dual program

$$\max\{v_1\gamma - v_2 | v_1 \geq 0; v_2 \geq 0; kv_1 - v_2 \leq -e^{-sk}, 0 \leq k \leq d\}.$$

The solution $(v_1, v_2) = (\frac{1}{d}(1-e^{-sd}), 1)$ gives rise to the same value as in (18). Hence, to prove optimality it suffices to prove that this solution is feasible. For this we need to show that

$$\frac{k}{d}(1-e^{-sd}) - 1 \leq -e^{-sk}, \;\; 0 \leq k \leq d.$$

This is trivially true for $k=0$ and for $k \geq 1$ this is equivalent to

$$\frac{1}{sd}(1-e^{-sd}) \leq \frac{1}{sk}(1-e^{-sk}), \;\; 1 \leq k \leq d.$$

The claim now follows since $\frac{1-e^{-x}}{x}$ is a decreasing function in x for $x \geq 0$.

We get

$$\begin{aligned}
\mathbb{E}[e^{-s\sum_{i=1}^{\mathrm{t}} X_i}] &= \mathbb{E}[e^{-s\sum_{i=i}^{\mathrm{t}-1} X_i}\mathbb{E}[e^{-sX_{\mathrm{t}}}|X_1, \cdots, X_{\mathrm{t}-1}]] \\
&\leq \left(1-\frac{\gamma}{d}(1-e^{-sd})\right)\mathbb{E}[e^{-s\sum_{i=1}^{\mathrm{t}-1} X_i}] \\
&\leq \left(1-\frac{\gamma}{d}(1-e^{-sd})\right)^{\mathrm{t}}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\Pr\{\sum_{i=1}^{\mathrm{t}} X_i \leq \alpha\gamma\,\mathrm{t}\} &\leq e^{s\alpha\gamma\,\mathrm{t}}\left(1-\frac{\gamma}{d}(1-e^{-sd})\right)^{\mathrm{t}} \\
&= e^{s\alpha\gamma\,\mathrm{t} + \mathrm{t}\ln\left(1-\frac{\gamma}{d}(1-e^{-sd})\right)} \\
&\leq e^{-\mathrm{t}\,\frac{\gamma}{d}(1-e^{-sd}-sd\alpha)}.
\end{aligned}$$

The desired inequality now follows by choosing $s = \frac{1}{d}\ln\frac{1}{\alpha}$.

Next we show that $\Pr\{\mathrm{T} < \infty | \, \mathrm{T} \geq \mathrm{t}\} \leq \frac{\xi^t}{1-\xi}$. The proof will be very similar to the preceding one and we will be short.

$$
\begin{aligned}
\Pr\{\mathrm{T} < \infty | \, \mathrm{T} \geq \mathrm{t}\} &= \Pr\{\bigcup_{i\geq 1}\{Y_i = 0\}| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&\leq \sum_{i=\mathrm{t}}^{\infty} \Pr\{Y_i = 0| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&= \sum_{i=\mathrm{t}}^{\infty} \Pr\{\sum_{k=1}^{i} X_k = (i-1)| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&\leq \sum_{i=\mathrm{t}}^{\infty} \Pr\{\sum_{k=1}^{i} X_k \leq i| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&= \sum_{i=\mathrm{t}}^{\infty} \Pr\{-\sum_{k=1}^{i} X_k \geq -i| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&= \sum_{i=\mathrm{t}}^{\infty} \Pr\{e^{-s\sum_{k=1}^{i} X_k} \geq e^{-si}| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}\} \\
&\leq \sum_{i=\mathrm{t}}^{\infty} e^{si}\mathbb{E}[e^{-s\sum_{k=1}^{i} X_k}| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}].
\end{aligned}
$$

Now for $i \geq \mathrm{t}$ we have

$$
\begin{aligned}
&\mathbb{E}[e^{-s\sum_{k=1}^{i} X_k}| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}] \\
\leq\;& \Big(1 - \frac{\gamma}{d}(1-e^{-sd})\Big)^{i-t+1} \mathbb{E}[e^{-s\sum_{k=1}^{t-1} X_k}| \bigcap_{j=1}^{\mathrm{t}-1}\{Y_j \geq 1\}] \\
\leq\;& \Big(1 - \frac{\gamma}{d}(1-e^{-sd})\Big)^{i-t+1} \mathbb{E}[e^{-s\sum_{k=1}^{\mathrm{t}-2} X_k}\mathbb{E}[e^{-sX_{\mathrm{t}-1}}|X_1, \cdots X_{\mathrm{t}-2}, X_{\mathrm{t}-1} \geq (\mathrm{t}-1) - \sum_{l=1}^{\mathrm{t}-2} X_l]| \bigcap_{j=1}^{\mathrm{t}-3}\{Y_j \geq 1\}].
\end{aligned}
$$

But

$$
\begin{aligned}
& \max_{u_k}\{\mathbb{E}[e^{-sX_i}|X_1,\cdots,X_{i-1},X_i \geq i-\sum_{l=1}^{i-1}X_l]|\mathbb{E}[X_i|X_1,\cdots,X_{i-1}]\geq\gamma; X_i\in\{0,\cdots,d\}\} \\
\leq\ & \max_{c,u_k}\{\frac{\sum_{k=c}^{d}e^{-sk}u_k}{\sum_{k=c}^{d}u_k}|u_k\geq 0;\sum_{k=0}^{d}ku_k\geq\gamma;\sum_{k=0}^{d}u_k=1\} \\
\leq\ & \max_{c,u_k}\{\sum_{k=c}^{d}e^{-sk}u_k|u_k\geq 0;\sum_{k=c}^{d}ku_k\geq\gamma;\sum_{k=c}^{d}u_k=1\} \\
\leq\ & \max_{u_k}\{\sum_{k=0}^{d}e^{-sk}u_k|u_k\geq 0;\sum_{k=0}^{d}ku_k\geq\gamma;\sum_{k=0}^{d}u_k=1\} \\
=\ & -\min_{u_k}\{-\sum_{k=0}^{d}e^{-sk}u_k|u_k\geq 0;\sum_{k=0}^{d}ku_k\geq\gamma;-\sum_{k=0}^{d}u_k\geq -1\} \\
=\ & 1-\frac{\gamma}{d}(1-e^{-sd}),
\end{aligned}
$$

so that

$$
\mathbb{E}[e^{-s\sum_{k=1}^{i}X_k}|\bigcap_{j=1}^{\mathrm{t}-1}\{Y_j\geq 1\}]\leq\Big(1-\frac{\gamma}{d}(1-e^{-sd})\Big)^i=\xi^i
$$

It follows that

$$
\begin{aligned}
\Pr\{\mathrm{T}<\infty|\,\mathrm{T}\geq\mathrm{t}\} &\leq \sum_{i=\mathrm{t}}^{\infty}\xi^i \\
&= \frac{\xi^t}{1-\xi}.
\end{aligned}
$$

To prove that for any $\mathrm{t}\geq 0$, $\Pr\{\mathrm{T}>\mathrm{t}\}>0$ note that from $\mathbb{E}[X_i|X_1,\cdots,X_{i-1}]\geq\gamma$ and $X_i\in\{0,\cdots,d\}$ we conclude that $\Pr\{X_i\geq 1|X_1\cdots,X_{i-1}\}\geq\frac{\gamma}{d}$. Therefore,

$$
\begin{aligned}
\Pr\{\mathrm{T}>\mathrm{t}\} &= \Pr\{Y_1\geq 1,\cdots,Y_{\mathrm{t}}\geq 1\} \\
&\geq \Pr\{X_1\geq 1,\cdots,X_{\mathrm{t}}\geq 1\} \\
&= \prod_{i=1}^{\mathrm{t}}\Pr\{X_i\geq 1|X_1\geq 1,\cdots,X_{i-1}\geq 1\} \\
&\geq \left(\frac{\gamma}{d}\right)^{\mathrm{t}}.
\end{aligned}
$$

It remains to prove that $\Pr\{\mathrm{T} < \infty\} \leq \mathrm{p}$. Note that for any $\mathrm{t} \geq 1$ we have

$$
\begin{aligned}
\Pr\{\mathrm{T} < \infty\} &= 1 - \Pr\{\mathrm{T} = \infty\} \\
&= 1 - \Pr\{\mathrm{T} = \infty | \, \mathrm{T} > \mathrm{t}\}\Pr\{\mathrm{T} > \mathrm{t}\} - \Pr\{\mathrm{T} = \infty | \, \mathrm{T} \leq \mathrm{t}\}\Pr\{\mathrm{T} \leq \mathrm{t}\} \\
&= 1 - \Pr\{\mathrm{T} = \infty | \, \mathrm{T} > \mathrm{t}\}\Pr\{\mathrm{T} > \mathrm{t}\} \\
&\leq 1 - \frac{1 - \xi - \xi^{\mathrm{t}}}{1 - \xi} \left(\frac{\gamma}{d}\right)^{\mathrm{t}}.
\end{aligned}
$$

□

C No Small Stopping Sets

Given a bipartite graph G with constraint set C we define a *stopping set* S to be any subset of C with the property that there does not exist a variable node in G with exactly one edge into S. The union of any two stopping sets is a stopping set. Hence, there exists a unique maximal stopping set. Thus, if the graph G is operated on by the diagonal extension stage of an approximate upper triangulation algorithm which looks for degree one variable nodes, then it always terminates with the residual graph determined by the maximal stopping set. I.e., after a diagonal extension stage, the residual graph consists of the constraint nodes in S, the maximal stopping set, the edges emanating from S and the variable nodes incident to these edges. Thus, one can show that the diagonal extension step will not terminate prematurely if one can show that there are no stopping sets.

Lemma 6. [No Small Stopping Sets] Let $\mathcal{C}^n(\lambda, \rho)$ be an ensemble of LDPC codes with $\underline{\mathrm{d}}_{\mathsf{c}} > 2$. Then there exist a positive number α and a natural number $n(\alpha)$ such that for all $n \geq n(\alpha)$ a randomly chosen element of $\mathcal{C}^n(\lambda, \rho)$ has probability at most $2^{-\sqrt{n}}$ of containing a stopping set of size in the range $[\sqrt{n}, \alpha n]$.

Proof. Recall the following simple estimates which we will use frequently in the sequel:

$$
\frac{1}{n+1} 2^{nh\left(\frac{k}{n}\right)} \leq \binom{n}{k} \leq 2^{nh\left(\frac{k}{n}\right)}
$$

as well as the fact that $h(x) = x \log \frac{1}{x} + O(x)$, for $0 \leq x \leq 1$.

Recall that the total number of edges is equal to $n \, \mathrm{d}_{\mathsf{v}}^{\mathrm{avg}}$. Consider e edges. More precisely, fix e check node sockets and for each such check node socket choose a variable node socket at random. Clearly, this can be done in

$$
\binom{n \, \mathrm{d}_{\mathsf{v}}^{\mathrm{avg}}}{e} e! \geq \frac{1}{n+1} 2^{n \, \mathrm{d}_{\mathsf{v}}^{\mathrm{avg}} h\left(\frac{e}{n \, \mathrm{d}_{\mathsf{v}}^{\mathrm{avg}}}\right)} e! = 2^{nh\left(\frac{e}{n}\right) + O(e)} e!
$$

ways. We will say that the e edges are *doubly connected* if each variable is either not connected at all or connected at least twice (with respect to these e edges). We claim that there are at most

$$\binom{n}{e/2}\binom{e/2\,\overline{\mathtt{d}}_{\mathtt{v}}}{e}e! \leq 2^{n(h(\frac{e}{2n})+\frac{e\,\overline{\mathtt{d}}_{\mathtt{v}}}{2n}h(\frac{2}{\overline{\mathtt{d}}_{\mathtt{v}}}))}e! = 2^{n\frac{1}{2}h(\frac{e}{n})+O(e)}e!$$

doubly connected constellations. To see this claim note that a doubly connected constellation with e edges involves at most $e/2$ variable nodes. Therefore, we get an upper bound if we count as follows: first choose $e/2$ out of the n variable nodes. These chosen variable nodes have at most $e/2\,\overline{\mathtt{d}}_{\mathtt{v}}$ sockets. Choose e of these sockets and connect to them the edges in any order. From this argument we see that the probability that the chosen set of *edges* is doubly connected is upper bounded by

$$2^{-(n\frac{1}{2}h(\frac{e}{n})+O(e))}. \tag{19}$$

Note that for sufficiently small e, (19) is *decreasing* in e.

Now consider a set of c check nodes. There are

$$\binom{n(1-r)}{c} \leq 2^{n(1-r)h\left(\frac{c}{n(1-r)}\right)} = 2^{nh(\frac{c}{n})+O(c)}$$

such sets. Each such set has at least $c\,\underline{\mathtt{d}}_{\mathtt{c}}$ edges and therefore assuming that c is sufficiently small we see from equation (19) that the probability that one such set is a stopping set is at most

$$2^{-(n\frac{\underline{\mathtt{d}}_{\mathtt{c}}}{2}h(\frac{c}{n})+O(c))}.$$

By the union bound it follows that the probability that a randomly chosen element of $\mathcal{C}^n(\lambda,\rho)$ does have a stopping set of size c is upper bounded by

$$2^{-(\frac{\underline{\mathtt{d}}_{\mathtt{c}}-2}{2}nh(\frac{c}{n})+O(c))} = 2^{-f(n,c,\underline{\mathtt{d}}_{\mathtt{c}})},$$

where we defined

$$f(n,c,\underline{\mathtt{d}}_{\mathtt{c}}) := \frac{\underline{\mathtt{d}}_{\mathtt{c}}-2}{2}nh\left(\frac{c}{n}\right)+O\left(c\right).$$

It remains to show that there exist constants $\alpha>0$ and $n(\alpha)>0$ such that for all $c\in[\sqrt{n},\alpha n]$ and for all $n\geq n(\alpha)$ we have $f(n,c,\underline{\mathtt{d}}_{\mathtt{c}})\geq\sqrt{n}$. Recall that $\underline{\mathtt{d}}_{\mathtt{c}}>2$. Therefore, if $\alpha>0$ and $\beta\in(\frac{1}{2},1)$ then

$$\begin{aligned}
f(n,\alpha n^\beta,\underline{\mathtt{d}}_{\mathtt{c}}) &= \frac{\underline{\mathtt{d}}_{\mathtt{c}}-2}{2}\alpha n^\beta\left(\log_2\left(\frac{n^{1-\beta}}{\alpha}\right)+O(1)\right)\\
&\geq \frac{\underline{\mathtt{d}}_{\mathtt{c}}-2}{2}\alpha n^\beta\\
&\geq \sqrt{n},
\end{aligned}$$

where the second step is true if we choose α small enough and the third step is true for sufficiently large n. $\square$