Protograph LDPC Codes with Block Thresholds: Extension to Degree-1 and Generalized Nodes

Asit Kumar Pradhan and Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras, Chennai 600036, India
Email: asit.pradhan,andrew@ee.iitm.ac.in

Abstract

Protograph low-density-parity-check (LDPC) are considered to design near-capacity low-rate codes over the binary erasure channel (BEC) and binary additive white Gaussian noise (BIWGN) channel. For protographs with degree-one variable nodes and doubly-generalized LDPC (DGLDPC) codes, conditions are derived to ensure equality of bit-error threshold and block-error threshold. Using this condition, low-rate codes with block-error threshold close to capacity are designed and shown to have better error rate performance than other existing codes.

I. INTRODUCTION

Low-density-parity-check (LDPC) codes [1], introduced by Gallager in 1960s, became popular in 1990s, because of their excellent performance under iterative message-passing decoding [2]. Several applications including security applications like wiretap coding [3] need code sequences with provable block-error threshold, i.e for any channel parameter below capacity, block-error rate should provably approach zero as blocklength goes to infinity. Two early efforts in block-error threshold for LDPC codes include consideration of ML decoding [1] and a weight enumeration analysis [4]. In [5], authors have shown that block-error and bit-error thresholds are same under message-passing decoder for codes having minimum variable node degree greater than two.
In our earlier work [6], we have extended the block-error condition in [5] allowing degree-two nodes in protograph LDPC ensembles under the condition that the degree-two subgraph of the protograph is a tree. In [6], we have also designed high rate codes (rate $\geq 1/2$) with block-error threshold close to capacity using differential evolution. However, computer search shows that low-rate codes (rate $\leq 1/3$) satisfying block-error threshold condition derived in [6] have threshold away from capacity. For example, a $7 \times 8$ optimized protograph defining a rate-1/8 code has a gap of 0.3 between block-error threshold and capacity over the Binary Erasure Channel (BEC). Low rate codes with bit-error threshold close to capacity have a large fraction of degree-one bit nodes [7] [8], which are not allowed by the block-error threshold condition in [6]. In this work, we extend the block-error threshold condition in [6] to allow degree-one bit nodes, which play an important role in designing low-rate codes with threshold close to capacity. Using the new block-error threshold condition, we have designed low-rate codes with block-error threshold close to capacity. For example, we have designed a rate-1/8 protograph LDPC code for BEC with a block-error threshold of 0.866 (gap of 0.009 from capacity). For the binary additive white Gaussian noise (BIAWGN) channel, we have designed a code of rate-1/4 and blocklength-64800 which has better BER/FER performance than the rate-1/4 code in the DVBS2 standard [9].

We also extend the block-error threshold condition to protograph Doubly Generalized LDPC (DGLDPC) codes, introduced in [10]–[12] and studied in [13]–[16]. Block-error condition for protograph DGLDPC ensembles allows cycles of degree-2 nodes and enables better optimization of codes.

Rest of the paper is organized as follows. Section II introduces definition and notation for protograph DGLDPC codes and describes density evolution over BEC and BIAWGN channel. Section III derives conditions on protograph and component codes under which block-error threshold equals density evolution threshold for large-girth ensembles. Optimization of protograph DGLDPC code is described in Section IV.

II. DEFINITIONS AND PRELIMINARIES

A general block-error threshold condition will be derived for doubly-generalized low density parity check (DGLDPC) codes. We will first define these codes formally and introduce notation for protograph DGLDPC codes.
A. Protograph DGLDPC codes

Protograph LDPC codes are defined by Tanner graphs that are created from a small base graph, called protograph, by a copy-permute operation. Protograph DGLDPC codes are defined in a similar way by allowing the variable and check nodes of the protograph to enforce arbitrary linear codes as component codes. The copy-permute operation for lifting the protograph remains the same.

Fig. 1 is an example of a protograph that expands to a DGLDPC code. The variable node \( v_4 \) and the check node \( c_1 \) have a (5,2) linear code as component code. All other component codes are standard. In the copy operation, all copies of \( v_4 \) and \( c_1 \) will retain the respective component codes.

A protograph is denoted \( G = (V \cup C, E) \), where \( V \) and \( C \) are the set of variable and check nodes, respectively, and \( E \) is the set of undirected edges. Multiple parallel edges are allowed between a variable node and a check node in a protograph. The edges connected to a variable node \( v_i \) or a check node \( c_j \) are denoted by \( e_{v_i,m} \) and \( e_{c_j,n} \), respectively, where \( m \in \{1, 2 \cdots d_{v_i}\} \) and \( n \in \{1, 2 \cdots d_{c_j}\} \). If \( v_i \) is connected to \( c_j \), then \( e_{v_i,m} = e_{c_j,n} \) for some \( m, n \).

The lifted or expanded graph is obtained by the copy-permute operation, and is specified by the number of copies and a permutation for each edge type. The design rate of the lifted graph is the same as that of the protograph. For the lifted graph, the sequence of component codes and edge types in the computation graph is completely determined by the protograph \( G \). This makes protograph density evolution analysis on girth-\( g \) liftings accurate up to iteration \( g/2 \).
Iterative message-passing decoding of DGLDPC codes is a generalized version of iterative decoding used for standard LDPC codes. At variable node $v_i$ enforcing a $(d_{v_i}, k_{v_i})$ component code with generator matrix $G$, channel information for $k_{v_i}$ bits is combined with incoming messages from check nodes in the previous iteration by extrinsic Maximum A Posteriori (MAP) processing on the extended component code with generator matrix $[G|I_{k_{v_i}}]$, where $I_{k_{v_i}}$ is the $k_{v_i} \times k_{v_i}$ identity matrix. At check node $c_j$, extrinsic MAP processing is performed using the enforced $(d_{c_j}, k_{c_j})$ component code.

B. Density evolution over BEC

Consider iterative message passing decoding on the lifted Tanner graph $G'$ derived from a DGLDPC protograph $G = (V \cup C, E)$ after transmission over a BEC with erasure probability $\epsilon$. In iteration $t$, let $x^t_{v_i,m}$ denote the probability that an edge of type $e_{v_i,m}$ carries an erasure from variable node to check node. Since the lifted graph has $|E|$ edge types, density evolution is a vector recursion that proceeds by computing $x^{t+1}_{v_i,m}$ for, $1 \leq i \leq |V|$, $1 \leq m \leq d_{v_i}$, from the vector $\{x^t_{v_i,m}\}$. Let $y^t_{c_j,n}$ denote the probability that an edge of type $e_{c_j,n}$ carries an erasure from check node to variable node in the $t$-th iteration.

Since MAP processing is done with the component code at check and variable nodes, the evolution of $x^t$ and $y^t$ will depend on the extrinsic messages generated by MAP decoders of the component codes. For obtaining an explicit expression for the probability of erasure of an extrinsic message generated by the MAP decoder of a linear code, a method based on the support weights and information functions of the linear code is used as described and discussed in [17]. An example of a (5,2) linear code, worked out in [17], is reproduced here and used later in an illustrative example of density evolution.

Example 1. Consider the (5,2) linear code with codewords $\{00000, 01011, 10101, 11110\}$. Let $x_i$, $i = 1, 2, \ldots, 5$, denote the independent input erasure probabilities of the 5 bits. Let $h_i(x_{\sim i})$, where $x_{\sim i} = \{x_1, \ldots, x_5\} \setminus \{x_i\}$ is the list of all variables except $x_i$, denote the output erasure probability of bit $i$. From [17], $h_i$ can be explicitly written in terms of $x_i$. For example, $h_1$ and $h_3$ are as follows:

\[
\begin{align*}
h_1(x_2, x_3, x_4, x_5) &= x_3 x_5 + x_2 x_3 x_4 - x_3 x_4 x_5 x_2, \\
h_3(x_1, x_2, x_4, x_5) &= x_1 x_5 + x_1 x_2 x_4 - x_1 x_4 x_5 x_2. \tag{1}
\end{align*}
\]
To proceed further, we assume that the extrinsic message-erasure probabilities from MAP processing at \( m \)-th edge of node \( v_i \) and \( n \)-th edge of node \( c_j \) have been derived, and these are denoted as \( h_{v_i,m}(\cdot) \) and \( h_{c_j,n}(\cdot) \), respectively. With this notation, the protograph density evolution recursion is given by the following equations for \( 1 \leq i \leq |V|, 1 \leq m \leq d_{v_i}, 1 \leq j \leq |C|, 1 \leq n \leq d_{c_j} \):

\[
x_{v_i,m}^0 = h_{v_i,m}(1_{d_{v_i}-1}, \epsilon 1_k), \\
y_{c_j,n}^{t+1} = h_{c_j,n}(x_{c_j,\sim n}^t), \\
x_{v_i,m}^{t+1} = h_{v_i,m}(y_{v_i,\sim m}^t, \epsilon 1_k).
\]

where \( x_{c_j,\sim n}^t = \{x_{c_j,1}, \ldots, x_{c_j,\sim n-1}, x_{c_j,n+1}, \ldots, x_{c_j,d_{c_j}}\} \), \( y_{v_i,\sim m}^t = \{y_{v_i,1}, \ldots, y_{v_i,\sim m-1}, y_{v_i,m+1}, \ldots, y_{v_i,d_{v_i}}\} \) and \( 1_k \) is the length-\( k \) all-ones vector. In the first iteration, shown in (2), the probability that an incoming message from a check node is an erasure is set as 1. Erasure probability from the channel is set to be \( \epsilon \).

**Example 2.** In Fig. [7] let the component codes at the variable node \( v_4 \) and check node \( c_1 \) be the \((5,2)\) code considered in Example [7]. All other variable and check nodes enforce repetition codes and SPC codes, respectively. As mentioned earlier, at \( v_4 \) MAP decoding is done over the extended version of the \((5,2)\) code with codewords \( \{0000000, 0101101, 1010110, 1111011\} \). Before starting recursion, assign \( y_{v_i,m}^0 = 1 \) for \( 1 \leq i \leq |V| \) and \( 1 \leq m \leq d_{v_i} \). The evolution for a few edges is shown below:

\[
x_{v_1,1}^{t+1} = \epsilon y_{v_1,2}^t, \quad x_{v_2,1}^{t+1} = \epsilon y_{v_2,1}^t, \\
x_{v_4,1}^{t+1} = \epsilon y_{v_4,3}^t y_{v_4,5}^t + \epsilon^2 y_{v_4,2}^t y_{v_4,3}^t y_{v_4,4}^t y_{v_4,5}^t, \\
y_{c_1,1}^t = x_{c_1,3}^t x_{c_1,5}^t + x_{c_4,2}^t x_{c_4,3}^t x_{c_4,1}^t - x_{c_1,2}^t x_{c_1,3}^t x_{c_1,4}^t x_{c_1,5}^t.
\]

### III. BLOCK-ERROR THRESHOLD EXTENSIONS

In this section, we generalize block-error threshold conditions for protograph LDPC codes to protograph with degree-1 variable nodes and to DGLDPC codes. The density evolution threshold, denoted \( \epsilon_{th} \), for the protograph ensemble is defined as the supremum of the set of \( \epsilon \) for which erasure probability on each edge tends to zero as iterations tend to infinity, i.e

\[
\epsilon_{th} = \sup \{ \epsilon : \max_{e \in E} x^t(e) \to 0 \}.
\]
Let us define $x^t$ as follows:

$$x^t = \sup_{e \in E} x^t(e).$$

In [6], sufficient conditions for block-error threshold being equal to bit-error threshold have been derived using the following two steps:

1. In first step, it has been shown that $x^t$ falls double exponentially with $t$, i.e.

   $$x^t = O(\exp(-\beta 2^{\alpha t}))$$

   with $\alpha > 0$, $\beta > 0$, when the degree-two subgraph of the protograph is a tree and $\epsilon \leq \epsilon_{\text{th}}$.

2. Under the assumption that girth, denoted by $g$, of the lifted code of blocklength $n$ is $O(\log n)$ and $t < g/2$, it has been shown that block-error threshold is same as bit-error threshold by upper bounding probability of block error, denoted by $P_B$, with $nx^t$ and using double-exponential fall property of $x^t$ as follows:

   $$P_B \leq nx_t = O(n \exp(-\beta 2^{\alpha t})) = O(n \exp(-\beta n^\alpha)).$$

The basic idea in the proof of step one is the following: When the degree-two subgraph of a protograph is a tree, variable node with degree greater than two is traversed in every $|V|$ (number of variable nodes) iterations of density evolution, resulting in squaring of $x^t$, which is sufficient to show double exponential fall of $x^t$ as described in [6]. Let us consider the computation graph
in Fig. 2 with a degree-one variable node \( v_2 \) and a degree-2 variable node \( v_1 \). In iteration \( t \), we have

\[
x^t_{e_1} = \epsilon y^t_{e_2} \\
= \epsilon(1 - (1 - x^{t-1}_{e_3})(1 - x^{t-1}_{e_4})(1 - x^{t-1}_{e_5})) \\
\geq \epsilon(1 - (1 - x^{t-1}_{e_3})) = \epsilon^2.
\] (5)

By using similar argument as above, it can also be shown that \( y^t \) corresponding to \( e_4 \) and \( e_5 \) is less than \( \epsilon \). This shows that the argument for double exponential fall as described in [6] does not carry over directly when the protograph has degree-one variable nodes even for edges that are not directly connected to degree-1 variable nodes.

A. Protograph LDPC code with degree-1 variable nodes

We say a function \( f(t) \) falls double exponentially with \( t \) if \( f(t) = O(\exp(-\beta 2^{\alpha t})) \) for sufficiently large \( t \), with \( \alpha \) and \( \beta \) being positive constants. The property of block-error threshold being equal to bit-error threshold does not require \( x^t_e \) and \( y^t_e \) for all \( e \in E \) to fall double exponentially. It is enough to show that probability of bit-error \( P_b \) corresponding to information bits falls double exponentially with iteration. Let \( P_b(v) \) be the probability of bit error corresponding to a variable node \( v \). We observe that \( P_b(v) \) falls double exponentially if \( y^t_e \) corresponding to at least one of incoming edges from a check node connected to \( v \) falls double exponentially. If \( x^t_e \) falls double exponentially with \( t \), then the edge \( e \) might help in the double exponential fall of \( y^t \) corresponding to other edges of protograph. To find out the set of variable nodes for which \( P_b(v) \) falls double exponentially with \( t \), we need to find out set of edges for which \( x^t_e \) and/or \( y^t_e \) fall double exponentially with \( t \).

Consider a protograph \( G(V \cup C, E) \). Let \( V_1 \subset V \) be the set of degree-one variable nodes and \( E_1 \subset E \) be the set of edges incident on them. Define \( C_1 = \{ c : c \text{ is connected to } v \in V_1 \} \). Let \( G_2 \) be the subgraph of \( G \) induced by degree-two variable nodes. Let \( E_2 \subset E \) be the set of edges of cycles in \( G_2 \). For a subgraph \( \hat{G}(\hat{V} \cup \hat{C}, \hat{E}) \) of the protograph \( G \), similarly define \( \hat{V}_1, \hat{C}_1, \hat{G}_2 \) and \( \hat{E}_2 \). For \( v \in \hat{V} \), let \( E_v \) and \( \hat{E}_v \) denotes the set of edges connected to \( v \) in protograph \( G \) and its subgraph \( \hat{G} \), respectively. Similarly, define \( E_c \) and \( \hat{E}_c \) for \( c \in \hat{C} \). Define \( D_y = \{ e : y^t_e \text{ falls double exponentially with } t \} \), \( D_x = \{ e : x^t_e \text{ falls double exponentially with } t \} \), \( \overline{D}_x = E \setminus D_x \), \( \overline{D}_y = E \setminus D_y \), \( D_{xy} = D_x \cap D_y \), \( \overline{D}_{xy} = E \setminus D_{xy} \). In the following two lemmas, we describe edges which are in \( \overline{D}_x, \overline{D}_y \) and \( \overline{D}_{xy} \).
Lemma 1. In the notation introduced above, the following are true:

1) $E_1 \subseteq \overline{D}_x$.
2) For $e \in E_1$ and $e' \in E_{ce} \setminus e$, $\{E_{ce} \setminus e\} \subseteq \overline{D}_y$.
3) $E_2 \subseteq \overline{D}_{xy}$.
4) If $e \in E_2$, then $E_{ce} \subseteq \overline{D}_y$.

Proof. 1) Consider $e \in E_1$. From (4), it follows that $x_e^t = \epsilon$ for all $t$. So, $E_1 \subseteq \overline{D}_x$.

2) Consider $e \in E_1$. Observe that for each $e' \in \{E_{ce} \setminus e\}$, $e \in \{E_{ce} \setminus e'\}$. From (3), it follows that for each $e' \in \{E_{ce} \setminus e\}$

$$y^{t'}_{e'} = 1 - \prod_{e'' \in E_{ce} \setminus e'} (1 - x_e^{t'-1})$$

$$\geq x_e^{t-1}.$$

Since $e \in E_1$, we know from Part 1, $x_e^{t-1} = \epsilon$. So, $y^{t'}_{e'} \geq \epsilon$ and $e' \in \overline{D}_y$.

3) Consider a cycle $L = e_1 e_2 \cdots e_l e_1$, with $v_{e_i} = v_{e_{i+1}}$. From (2)-(3), it follows that

$$x^{(t+l)}_{e_i} = \epsilon y^{(t+l)}_{e_{i+1}}$$

$$\geq \epsilon x^{(t+l-1)}_{e_{i+2}}.$$

By applying (2) and (3) $l/2$ times alternatively, it can be shown that $x^{(t+l)}_{e_i} \geq \epsilon^{\frac{l}{2}}(x^{t}_e)$ for $1 \leq i \leq l$. Similarly, it can be shown that $y^{(t+l)}_{e_i} \geq \epsilon^{\frac{l}{2}}(y^{t}_e)$. So, $e_i \in \overline{D}_{xy}$.

4) Consider $e \in E_2$. For each $e' \in E_{ce}$, observe that $\{E_{ce} \setminus e'\} \cap E_2 \neq \emptyset$. Let $\tilde{e} \in \{E_{ce} \setminus e'\} \cap E_2$.

From (2), it follows that for each $e' \in E_{ce}$

$$y^{t'}_{e'} = 1 - \prod_{e'' \in E_{ce} \setminus e'} (1 - x_e^{t'-1})$$

$$\geq x_e^{t-1}.$$

Since $\tilde{e} \in E_2$, we know from Part 3 of Lemma 1 that $x_{\tilde{e}}^{t-1}$ does not fall double exponentially with $t$. So, $y^{t'}_{e'}$, for $e \in E_2$ and $e' \in E_{ce}$, does not fall double exponentially with $t$.

Define $\hat{E} = E - \{E_1 \cup E_2\}$. Let $\hat{G}(\hat{V} \cup \hat{C})$ be the subgraph of $G$ induced by edges in $\hat{E}$. In context of double exponential fall, $\hat{E}_1$ and $\hat{E}_2$ behave in same way as $E_1$ and $E_2$, which will be shown in the following lemma.
Lemma 2. In the notation defined above

1) $\hat{E}_1 \subseteq \overline{D}_x$.

2) If $e \in \hat{E}_1$, then $\{E_{ce} \setminus e\} \subseteq \overline{D}_y$.

3) $\hat{E}_2 \subseteq \overline{D}_{xy}$.

4) If $e \in \hat{E}_2, E_{ce} \subseteq \overline{D}_y$.

Proof. 1) Consider $e \in \hat{E}_1$. We will use the fact that $x^t_e$ falls double exponentially iff $y^{t+1}_{e'}$ corresponding to at least one edge $e' \in E_{ce} \setminus e$ falls double exponentially. We have

$$x^t_e = \epsilon \prod_{e' \in E_{ve} \setminus e} y^{t}_{e'}$$

$$= \epsilon \prod_{e' \in (E_{ve} \setminus \hat{E}_{ve})} y^{t}_{e'} \prod_{e' \in \hat{E}_{ve} \setminus e} y^{t}_{e'}.$$  

Define $A^t_e = \prod_{e' \in E_{ve} \setminus \hat{E}_{ve}} y^{t}_{e'}$. From Lemma 1, it can be deduced that $A^t_e$ does not fall double exponentially with $t$. Since $e$ is incident on a degree-one variable node of $\hat{G}$, $\hat{E}_{ce} \setminus e = \emptyset$. So, $x^t_e = \epsilon A^t_e$ and it does not fall double exponentially with $t$.

2) Similar to the proof of Part 2 of Lemma 1.

3) Consider a cycle $\hat{L} = e_1 e_2 \cdots e_l e_1$ with $v_{e_i} = v_{e_{i+1}}$.

$$x^{t+l}_{e_i} = \epsilon \prod_{e \in E_{ve_i} \setminus v_i} y^{t+l}_{e}$$

$$= \epsilon \prod_{e \in (E_{ve_i} \setminus \hat{E}_{ve_i})} y^{t+l}_{e} \prod_{e \in \hat{E}_{ve_i} \setminus v_i} y^{t+l}_{e}$$

$$= A^{t+l}_{e} y^{t+l}_{e}$$

$$\geq A^{t+l}_{e} x^{t+l-1}_{e_{i+2}},$$

where $A^{t+l}_{e} = \prod_{e \in (E_{ve_i} \setminus \hat{E}_{ve_i})} y^{t+l}_{e}$ and $e_{i+1} = \hat{E}_{ve_i} \setminus e_i$. From Lemma 1, it can be deduced that $A^{t+1}_{e} x^{t+l}_{e_{i+2}}$ does not fall double exponentially. By applying above $l/2$ times we can show that

$$x^{t+l}_{e} \geq A^{t}_{e} x^{t}_{e_{i}}.$$  

Since $A^{t}_{e}$ does not fall double exponentially, $x^{t+l}_{e_{i}}$ does not fall double exponentially with $t$.

4) Similar to the proof of Part 4 of Lemma 1
1) **Subgraph RED(G):** The following algorithm finds a subgraph, denoted by $\text{RED}(G)$, such that edges of $\text{RED}(G)$ are in $D_{xy}$. This is done by recursively removing edges in $E_1$ and $E_2$ using Lemmas 1 and 2.

1) $\hat{G} = G(V \cup C, E)$.

   while $\hat{G}(\hat{V} \cup \hat{C}, \hat{E})$ has variable nodes of degree-one, or the subgraph induced by degree-two variable nodes is not a tree do

   2) $\hat{G}_2(\hat{V}_2 \cup \hat{C}_2, \hat{E}_2)$: Subgraph induced by degree-two variable node in $\hat{G}$. $V'_2 = \{v \in \hat{V}_2 : v$ belongs to a cycle in $\hat{G}_2\}$, $C'_2 = \{c \in \hat{C} : c$ is connected to some $v \in V'_2\}$

   3) $\hat{G} = \hat{G} - \{V'_2, C'_2\}$.

   4) $V'_1 = \{v \in \hat{V} : \text{deg}(v) = 1\}$, $C'_1 = \{c \in \hat{C} : c$ is connected to some $v \in V'_1\}$.

   5) $\hat{G} = \hat{G} - \{V'_1, C'_1\}$ (delete nodes and edges connected to them).

   end while

6) $\text{RED}(G) = \hat{G}$.

2) **Double Exponential Fall:** We will now show that $x^t$ and $y^t$ corresponding to each edge $e \in \text{RED}(G)$ fall double exponentially in the density evolution analysis of protograph $G$. For an edge $e \in E_{\text{RED}(G)}$, observe that $E_{c_e} \subset E_{\text{RED}(G)}$. So, from (3), it is easy to see that if $x^t_e$ falls double exponentially for all $e \in E_{\text{RED}(G)}$, then $y^t_e$ will fall double exponentially for all $e \in E_{\text{RED}(G)}$. So, it is enough to show $x^t_e$ for all $e \in \text{RED}(G)$ falls double exponentially with $t$.

**Theorem 1.** Let $E_{\text{RED}(G)}$ denote edges of $\text{RED}(G)$. Let $\overline{x'} = \max_{e \in E_{\text{RED}(G)}} x^t(e)$, where $x^t(e)$ is the erasure probability along edge $e$ in the density evolution recursion of $G$. Let $\epsilon_{in}$ denote the density evolution threshold of $G$. If $\text{RED}(G)$ is non-empty, then $E_{\text{RED}(G)} \subset D_{xy}$.

**Proof.** The proof follows the proof of [6, Theorem 1] very closely. We will briefly sketch the proof here.

First observe that $E_{\text{RED}(G)} \subset E$. Let us consider $e \in E_{\text{RED}(G)}$. From the density evolution
recursions in (2)-(4), we know
\[ y_{t,v}^{t+1} = C_2(x^{t}), \quad (6) \]
\[ x_{v,m}^{t+1} \leq C_1 \epsilon \left( \max_{m \in \{1, \ldots, d_v\}} y_{v,m}^{t+1} \right)^{d_v(G) - 1}, \quad (7) \]
\[ \leq C_1 \epsilon \left( \max_{m \in \{1, \ldots, d_v\}} y_{v,m}^{t+1} \right)^{d_v(RED(G)) - 1}, \quad (8) \]
where \( C_1 \) and \( C_2 \) are positive constants, \( d_v(G) \) and \( d_v(RED(G)) \) are degree of the variable node \( v_i \) in \( G \) and \( RED(G) \), respectively. By combining (6) and (8), we get
\[ x^{t+1} \leq \left\{ C_2 \right\}^{d_v(RED(G)) - 1} C_1 \left( (x^t) \right)^{d_v(RED(G)) - 1}. \quad (9) \]
Let \( v_2 \) be the number of degree-two variable nodes in \( RED(G) \). Now, a squaring term appears in the RHS of the bound above at least once in every \( 2v_2 + 1 \) iterations because in every edge traversal on \( RED(G) \) with \( 2v_2 + 1 \) steps there is at least one variable node with degree at least 3. Since \( \epsilon \leq \epsilon_{th} \), \( t \) can be chosen sufficiently large to make the constant term in the expression of \( x_t \) to be smaller than one. This can be formally shown using the same argument as in [6]. Therefore, we have proved that there exists \( R \) such that
\[ x^{t+2v_2+1} \leq A(x^t)^2, \]
where \( A \) is constant and \( A < 1 \) for \( t > R \). By applying the above repeatedly, we can show that
\[ x^{R+i(2v_2+1)} \leq A^{-1}(A^{R})^{2^i}, \quad (10) \]
for every positive integer \( i \), which implies \( E_{RED(G)} \subset D_{xy} \) and the proof of the theorem is complete. \( \square \)

In Theorem 1, we have shown that \( E_{RED(G)} \subset D_{xy} \). If \( e \in D_{xy} \), then \( x_{e'}^{t} \) corresponding to edge \( e' \in \{E_{e} \setminus e\} \) falls double exponentially. So, \( \{E_{e} \setminus e\} \subset D_{x} \) for each \( e \in E_{RED(G)} \). Now consider an edge \( e \notin E_{RED(G)} \). We have
\[ y_{e}^{t} = 1 - \prod_{e' \in E_{e} \setminus e} (1 - x_{e'}^{t-1}). \]
If \( \{E_{e} \setminus e\} \subset D_{x} \), then \( e \in D_{y} \). Using the above two steps, we will find \( D_y \) and \( D_x \) from \( E_{RED(G)} \) by using a message passing algorithm. Let \( r_{e}^{t} \) and \( s_{e}^{t} \) denote messages on edge \( e \) in
1) If \( e \in E_{RED(G)} \), then initialize \( r^0_e = 1 \), otherwise \( r^0_e = 0 \).

2) For \( e \in E \), if \( \exists e' \in \{E_{ve} \setminus e\} \) such that \( r^t_{e'} = 1 \), then \( s^t_e = 1 \).

3) For \( e \in E \), if \( s^t_e = 1 \) \( \forall e' \in \{E_{ce} \setminus e\} \), then \( r^{t+1}_e = 1 \).

4) Continue 2 and 3 till \( r^t_e = r^{t-1}_e \) and \( s^t_e = s^{t-1}_e \).

5) \( D_x = \{ e \in E : s^t_e = 1 \} \) and \( D_y = \{ e \in E : r^t_e = 1 \} \).

---

Fig. 3: \( RED(G) \) : Empty.

\( t \)-th iteration from check node to variable node and variable node to check node, respectively. Algorithm \ref{Alg1} and \ref{Alg2} are illustrated through the following examples.

**Example 3.** Consider the rate-1/4 LDPC protograph \( G \) in Fig. 3a. \( G \) has a degree-one variable node \( v_1 \). After removing \( v_1 \), and the check node \( c_1 \) connected to \( v_1 \) from \( G \), we get the reduced protograph shown in Fig. 3b. Removal of edges connected to \( v_1 \) and \( c_1 \) reduces degree of variable node \( v_2 \) to one. Removal of newly introduced degree-one variable node \( v_2 \) and check node connected to it introduces a cycle \( v_4c_4 \) formed by degree-two variable nodes. Removal of loop \( v_4c_4 \) from Fig. 3c results in an empty \( RED(G) \). So, \( D_{xy} = D_y = D_x = \emptyset \).

**Example 4.** Consider the rate-1/4 protograph \( (G) \) in Fig. 4a. After removal of degree-one variable node \( v_1 \) and its neighboring check node \( c_1 \), we get Fig. 4b. Removal of \( v_1 \) and \( c_1 \) introduces a degree-one variable node \( v_2 \) in Fig. 4b. After removal of \( v_2 \), we get Fig. 4c, which does not have a variable node with degree \( \leq 2 \). Hence, \( x^t \) and \( y^t \) for all edges in Fig. 4c have double exponential fall property in density evolution analysis of \( G \). Algorithm \ref{Alg1} is illustrated through Fig. 5. Algorithm \ref{Alg2} starts by assigning \( r^0_e = 1 \) for \( e \in E_{RED(G)} \) and \( r^0_e = 0 \) for \( e \in E \setminus E_{RED(G)} \). In Fig. 5, edges are labeled with messages carried by them. Arrow indicates the direction of message in an edge. After end of Algorithm \ref{Alg2} we get \( D_y = \{ E_{RED(G)}, v_2c_2, v_1c_1 \} \).
3) Extension to DGLDPC protograph: For an edge $e$ in a DGLDPC protograph $G$, let $h_{ce}$ and $h_{ve}$ denote the extrinsic message erasure probabilities at the check node and variable node, respectively. Note that $h_{ce}$ and $h_{ve}$ are polynomials in multiple variables denoting erasure probabilities of edges in $E_{ce} \setminus e$ and $E_{ve} \setminus e$, respectively (see (2)-(4)). Let $d_{ce}$ and $d_{ve}$ denote the least sum degree of terms in $h_{ce}$ and $h_{ve}$, respectively. In a standard protograph, edges from degree-1 variable nodes and edges in loops formed by degree-2 variable nodes do not have double exponential fall in erasure probabilities. In a DGLDPC protograph, the corresponding edges are as follows. Let $E_1 = \{ e : d_{ve} = 0 \text{ or } d_{ce} = 0 \}$, i.e $E_1$ is the set of edges with a constant term in their corresponding $h_{ce}$ or $h_{ve}$. Let $G_2$ be the subgraph induced by edges in $\{e : d_{ce} = 1 \text{ or } d_{ve} = 1\}$. A cycle $\{e_1e_2\cdots e_{2l}e_1\}$ is said to be non-DEX (does not have double exponential fall property) if the $2l$ multivariate polynomials $h_{ce_1}, h_{ve_2}, h_{ce_3}, h_{ve_4}, \cdots, h_{ce_{2l-1}}, h_{ve_{2l}}$ share at least one degree-1 term. Let $E_2$ be the set of edges in non-DEX cycles of $G_2$.

Algorithm \cite{22} is run with the above generalized definition of $E_1$ and $E_2$ to find RED($G$) for a DGLDPC protograph $G$. Another modification is as follows: after removing an edge, the
message erasure probability of all edges are updated by replacing the message corresponding to the removed edge by 1. Algorithm ?? is extended with no significant modification.

4) Block-error threshold: We now use Theorem ?? and its generalized version to a sequence of large girth liftings of a DGLDPC protograph $G$ and state conditions for block-error threshold property. Let us denote the set of variable nodes of $G$ for which $P_b(v)$ falls double exponentially by $\text{DEX}(V)$. Let $\overline{G}$ be the code lifted from protograph $G$ with blocklength $n$ and message length $k$. Let us define $\overline{P}_b$ as $\overline{P}_b = \max_{v \in V} P_b(v)$, where $V$ is a set of variable nodes corresponding to message bits. Probability of block error can be bounded as $P_B < k \overline{P}_b$. Let $V_D$ be the variable nodes in $\overline{G}$ corresponding to $\text{DEX}(V)$.

**Theorem 2.** In the notation introduced above, if $V_I \subset V_D$, and girth of $\overline{G}$ is at least $c \log n$, then

$$P_B \leq k \mathcal{O}(\exp(-\beta n^\alpha)),$$

where $\alpha, \beta, c$ are positive constants.

**Proof.** We know that $P_b(v)$ corresponding to $v \in \text{DEX}(V)$ fall double exponentially in density evolution of $G$, i.e.,

$$P_b(v) = \mathcal{O}(\exp(-\beta 2^\alpha t))$$

for $v \in \text{DEX}(V)$ with $\alpha, \beta$ being positive constants. Since $V_I \subset V_D$, $P_b(v) = \mathcal{O}(\exp(-\beta 2^\alpha t))$ for $v \in V_I$. So, probability of block error of $\overline{G}$, denoted by $P_B$, can be bounded as follows:

$$P_B \leq k \mathcal{O}(\exp(-\beta 2^\alpha t))$$

for $\epsilon \leq \epsilon_{th}$. Assuming $t < g/2$ and putting $t = c \log n$, we get

$$P_B \leq k \mathcal{O}(\exp(-\beta n^\alpha)).$$

From above theorem, if $\epsilon < \epsilon_{th}$, we can deduce that $P_B \to 0$ as $n \to \infty$. The rate-$1/4$ protograph in Fig. 4a has one information bit and it satisfies the block-error threshold condition, because $P_b$ corresponding to degree-three variable node of $G$ fall double exponentially as described in Example 3. So, block-error threshold and bit-error threshold can be made equal for appropriate lifting size using Theorem 2. Protographs can be lifted to have large girth ($\mathcal{O}(c \log n)$) by using the large girth construction in [6]. Similarly, the rate-$1/4$ protograph in Fig. 3a has
one information bit. However, for this protograph, block-error threshold cannot be made equal to bit-error threshold by using Theorem 2 because \( P_b \) corresponding to any variable nodes of \( G \) does not fall double exponentially.

The block-error threshold condition for BIAWGN channel is same as block-error threshold condition for BEC and can be proved using a Bhattacharya parameter argument as in \([6, \text{Theorem 3}]\) and \([18, \text{Theorem 2}]\).

IV. OPTIMIZED DGLDPC PROTOGRAPHS

In this section, we design capacity-approaching protographs with block-error threshold by using the condition derived in Section III. Let \( G \) be a protograph of size \(|V| \times |C|\), where \( V \) and \( C \) denote the set of variable and check nodes. We divide variable nodes in \( V \) into two sets - standard variable nodes denoted \( V_s \), and generalized variable nodes denoted \( V_g \). \( C_s \) and \( C_g \) are similar notations for check nodes. We use repetition code and SPC code at standard variable nodes and check nodes, respectively. At a generalized node \( v \), we choose a \((d_v, k_v)\) linear code as component code. To design a rate-\( r \) code, we choose component codes at generalized nodes in such a way that \( r = 1 - \frac{\sum_i |C| (d_i - k_i)}{\sum_j |V| k_j} \). At standard variable node \( v \) and check node \( c \), we have \( k_v = 1 \) and \( d_c - k_c = 1 \). We maximize the threshold of protograph over the connections of protograph, degree of standard nodes, and label of edges connected to generalized nodes by using differential evolution \([19]\).

A. Differential Evolution

Different steps of differential evolution are elaborated as follows. The details of optimizing labels of edges at generalized nodes is skipped for brevity.

1) Initialization is done as follows

- Start with \(|C||V|\) base matrices \( B_{k,0} \), \( 0 \leq k \leq |C||V| \), each of size \(|C| \times |V|\). To restrict the search space, entries of base matrices are chosen randomly from the set \( \{1, 2, \cdots, 8\} \). Enforce variable and check node degree constraint at generalized nodes, i.e. \( \sum_{i=1}^{|C|} B_{k,0}(i, j) = d_{v_j}, \text{ for } v_j \in V_g, \sum_{j=1}^{|V|} B_{k,0}(i, j) = d_{c_i}, c_i \in C_g. \)

If \( B_{k,0} \) does not satisfy block-error threshold condition derived in Theorem 2 add an edge between degree-1 or degree-2 standard variable node and standard check node, chosen randomly. Continue adding such edges till the block-error threshold condition is satisfied.
2) Mutation: Protographs of generation $G$ ($G = 0, 1, \cdots$) are interpolated as follows.

$$M_{k,G} = [B_{r_1,G} + 0.5(B_{r_2,G} - B_{r_3,G})],$$

(11)

where $r_1, r_2, r_3$ are randomly-chosen distinct values, and $[x]$ denotes the absolute value of $x$ rounded to the nearest integer.

3) Crossover: The $(i, j)$-th entry of a candidate protograph $B'_{k,G}$ is set as the $(i, j)$-th entry of $M_{k,G}$ with probability $p_c$, or as the $(i, j)$-th entry of $B_{k,G}$ with probability $1 - p_c$. We use $p_c = 0.88$ in our optimization runs.

4) Selection: If the threshold of $B_{k,G}$ is greater than that of $B'_{k,G}$ and it satisfies block-error threshold condition in Theorem 2, set $B_{k,G+1} = B_{k,G}$; else, set $B_{k,G+1} = B'_{k,G}$.

5) Termination: Steps 2–4 are run for several generations (we run up to $G = 6000$) and the protograph that gives the best threshold is chosen as the optimized protograph.

We compute thresholds of protographs for the BEC by using the density evolution described in Section II-B. We compute thresholds of protograph for AWGN channel using the EXIT function method described in [20].

B. Lifting Protographs

A $|v| \times |c|$ protograph $G$ is lifted to a LDPC code of length $n$ as follows:

1. Let $m$ denote the maximum element of the corresponding $|V| \times |C|$ base matrix $B$. Replace $B(i, j)$ with a random $m \times m$ binary matrix $M$ with row and column sum equal to $B(i, j)$. If $B(i, j) = 0$, replace by an $m \times m$ zero matrix. Denote the new $|V|m \times |C|m$ binary matrix as $B'$.

2. We assume $m|V|$ divides $n$. Replace each nonzero entry of $B'$ with a $\frac{n}{|V|m} \times \frac{n}{|V|m}$, randomly generated, circular shift permutation matrix.

There can be more sophisticated liftings. However, we use the above simple method in all simulations and do not attempt to optimize the lifting further.

C. Optimized protographs for BEC

For BEC, optimized LDPC protographs (base matrices) of rate $1/10$ and $1/8$ with block-error thresholds within 0.01 of capacity are given in (14) and (15), respectively, in the Appendix. It is observed that optimized protographs have significant fraction of degree-one variable nodes.
Thresholds of LDPC protographs with degree-one nodes, LDPC protographs without degree-one nodes, GLDPC protographs with degree-one nodes and AR4A protograph [21] for BEC have been compared in Table I. We see that optimized protographs have better thresholds when degree-one nodes are allowed in optimization. For example, an optimized, rate-1/8, Protograph with degree-one bit nodes in (15) has threshold 0.866 over BEC, while optimized, rate-1/8 protograph without degree-one nodes has a threshold 0.85. From Table I, it is also observed that use of a generalized component code does not improve the threshold. However, generalized nodes are useful in designing smaller protographs with block-error threshold reasonably close to capacity. For example, an optimized $8 \times 10$, rate-1/8 DGLDPC protograph has block-error threshold 0.86 which is quite close to 0.866 achieved with a $21 \times 24$ LDPC protograph.

Optimized protographs in Table I are lifted to codes of blocklength 5000 using the method in Section IV-B and their BER/FER are simulated using the standard message-passing decoder. The plots are shown in Fig 6a. For comparison, AR3A/AR4A [7] protographs are lifted to the same blocklength of 5000 using the method in Section IV-B and their BER/FER are plotted in Fig. 6a. We see that the BER and FER of optimized codes are better than that of AR4A codes of same rate.

D. Optimized Protographs For AWGN

Observations similar to the BEC case hold for AWGN channel as well. Fig. 6(b) compares BER/FER of optimized rate-1/3 and rate-1/4 codes with codes of same rate from DVBS2 standard [9] and AR4A family. The blocklength for all codes are fixed to 64800. Parity check matrix corresponding to optimized protographs and AR4A protographs are obtained by method described in Section IV-B. BER and FER performance of optimized codes are obtained by method described in Section IV-B. BER and FER performance of optimized codes are better than the corresponding DVBS2/AR4A codes. BER/FER performance of optimized codes can be further improved by using more optimized liftings.

In summary, we designed low-rate codes with block-error threshold close to capacity. From simulation, we observe that optimized codes have better FER performance than comparable protographs of same rate.

REFERENCES

[1] R. Gallager, “Low-density parity-check codes,” Ph.D. dissertation, Massachusetts Institute of Technology, 1963.
[2] D. Mackay and N. Neal, “Near Shannon limit performance of low density parity check codes,” IEEE Electron. Letter, vol. 2, pp. 1645–1646, 1996.
(a) Performance of codes over BEC, length=5000.

(b) Performance of codes over AWGN, length=64800.

Fig. 6: BER:Solid, FER:Dashed.
| Rate | Size of Protograph | Types of Protograph | DE Threshold | Block Threshold |
|------|--------------------|---------------------|--------------|-----------------|
| 1/10 | $27 \times 30$ in [14] | LDPC with degree-1 | 0.894 | Yes |
|      | $10 \times 11$ in [7, Fig. 12] | AR4JA | 0.868 | No |
|      | $17 \times 23$ | GLDPC with degree-1 | 0.892 | Yes |
|      | $27 \times 30$ | LDPC w/o degree-1 | 0.877 | Yes |
| 1/8  | $21 \times 24$ in [15] | LDPC with degree-1 | 0.866 | Yes |
|      | $8 \times 9$ in [7, Fig. 11] | AR4JA | 0.846 | No |
|      | $13 \times 19$ | GLDPC with degree-1 | 0.866 | Yes |
|      | $14 \times 16$ in [6] | LDPC w/o degree-1 | 0.85 | Yes |
|      | $8 \times 10$ | DGLDPC w/o degree-1 | 0.86 | Yes |

**TABLE I: Optimized protographs and thresholds for BEC.**

[3] A. Subramanian, A. Suresh, S. Raj, A. Thangaraj, M. Bloch, and S. McLaughlin, “Strong and weak secrecy in wiretap channels,” in *Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on*, Sep. 2010, pp. 30–34.

[4] D. Divsalar, C. Jones, S. Dolinar, and J. Thorpe, “Protograph based LDPC codes with minimum distance linearly growing with block size,” in *GLOBECOM ’05. IEEE Global Telecommunications Conference, 2005.*, vol. 3, Nov 2005.

[5] M. Lentmaier, D. Truhachev, K. Zigangirov, and D. Costello, “An analysis of the block error probability performance of iterative decoding,” *IEEE Trans. Inf. Theory*, vol. 51, no. 11, pp. 3834–3855, Nov. 2005.

[6] A. Pradhan, A. Thangaraj, and A. Subramanian, “Construction of near-capacity protograph LDPC code sequences with block-error thresholds,” *IEEE Trans. Commun.*, vol. 64, no. 1, pp. 27–37, Jan 2016.

[7] D. Divsalar, S. Dolinar, and C. Jones, “Low-rate LDPC codes with simple protograph structure,” in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, Sept 2005, pp. 1622–1626.

[8] T. Richardson and R. Urbanke, “Multi-edge type LDPC codes,” Apr. 2004, unpublished. [Online]. Available: [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.7310](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.7310)

[9] “Digital video broadcasting; second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering,” *EN 302 307 V1.1.1*, 2004. [Online] Available: [http://www.etsi.org](http://www.etsi.org).

[10] R. Tanner, “A recursive approach to low complexity codes,” *Information Theory, IEEE Transactions on*, vol. 27, no. 5, pp. 533–547, Sep 1981.
| Rate | Size of Protograph | Types of Protograph | DE Threshold | Block Threshold |
|------|-------------------|---------------------|--------------|-----------------|
| 1/4  | $30 \times 40$ in [13] | LDPC with degree-1 | -0.630 | Yes |
|      |                   | DVBS2               | -0.35        | No |
|      | $4 \times 5$ in [7, Fig.8] | AR4A               | -0.522       | No |
| 1/3  | $30 \times 45$ in [12] | LDPC with degree-1 | -0.330       | Yes |
|      |                   | DVBS2               | -0.1         | No |
|      | $3 \times 4$ in [7, Fig.7] | AR4A               | -0.130       | No |

TABLE II: Optimized protographs and thresholds for AWGN.

[11] M. Lentmaier and K. Zigangirov, “On generalized low-density parity-check codes based on Hamming component codes,” *Communications Letters, IEEE*, vol. 3, no. 8, pp. 248–250, Aug 1999.

[12] J. Boutros, O. Pothier, and G. Zemor, “Generalized low density (Tanner) codes,” in *Communications, 1999. ICC '99. 1999 IEEE International Conference on*, vol. 1, 1999, pp. 441–445 vol.1.

[13] E. Paolini, M. Fossorier, and M. Chiani, “Doubly-generalized LDPC codes: Stability bound over the BEC,” *Information Theory, IEEE Transactions on*, vol. 55, no. 3, pp. 1027–1046, March 2009.

[14] Y. Wang and M. Fossorier, “Doubly generalized LDPC codes over the AWGN channel,” *Communications, IEEE Transactions on*, vol. 57, no. 5, pp. 1312–1319, May 2009.

[15] E. Paolini, M. Fossorier, and M. Chiani, “Generalized and doubly generalized LDPC codes with random component codes for the binary erasure channel,” *Information Theory, IEEE Transactions on*, vol. 56, no. 4, pp. 1651–1672, April 2010.

[16] S. Abu-Surra, D. Divsalar, and W. Ryan, “Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes,” *Information Theory, IEEE Transactions on*, vol. 57, no. 2, pp. 858–886, Feb 2011.

[17] M. Lentmaier, M. Tavares, and G. Fettweis, “Exact erasure channel density evolution for protograph-based generalized LDPC codes,” in *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, June 2009, pp. 566–570.

[18] A. K. Pradhan and A. Thangaraj, “Near-capacity protograph doubly-generalized LDPC codes with block thresholds,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2534–2538.

[19] A. Shokrollahi and R. Storn, “Design of efficient erasure codes with differential evolution,” in *Information Theory, 2000. Proceedings. IEEE International Symposium on*, 2000, p. 5.

[20] E. Sharon, A. Ashikhmin, and S. Litsyn, “EXIT functions for binary input memoryless symmetric channels,” *IEEE Transactions on Communications*, vol. 54, no. 7, pp. 1207–1214, July 2006.
APPENDIX A
OPTIMIZED CODES

Non zero entries of optimized base matrices corresponding to different rates are given below. Non zero entries of the $i$-th row of a base matrix is listed next to $i$. Superscript denotes the element at that location. If superscript is not mentioned, then non-zero element at that location is 1.

\begin{align*}
1 : 3, 10, 15^2, 32, 37, 39. & 2 : 8^2, 11, 12, 14, 25. & 3 : 8, 10, 12, 13, 25, 29, & 4 : 5, 8^2, 25, 39. \\
5 : 2, 4, 6, 8, 11, 26, 29, & 6 : 6, 8, 13, 15, 22, 25. & 33. & 7 : 1, 8^2, 10, 12, 32, \\
31, 40, 43, 45. & 8 : 11, 14, 18, 22, 23. & 9 : 3, 5, 9, 10, 19, 36, & 33, 35, 42, 43. \\
10 : 3, 6, 8^2, 15, 20, 29. & 25, 37. & 42, 44. & 11 : 6, 8, 23, 25. \\
12 : 1, 8, 10, 12, 25. & 13 : 8, 13^2, 14, 23, 25. & 14 : 8^2, 10, 13, 15, 34, & 15 : 3, 6, 8, 10, 11, 14, \\
26, 27. & 27, 42. & 25, 27, 31, 42. & 27, 31, 42. \\
16 : 5, 6, 8, 9, 15, 25, 40. & 17 : 5, 8, 23, 25, 43. & 18 : 2, 7, 8, 9, 15, 19, & 19 : 4, 5, 6, 8, 14, 25. \\
20 : 1, 6, 7, 9, 14, 26, & 21 : 2, 8, 12, 13, 25, 28, & 28, 37. & 22 : 5^2, 6, 17, 25, 43. \\
41. & 32, 41, 45. & 23 : 5, 8, 19, 23, 25, 37, & 24 : 3, 6, 8, 11, 19. \\
25 : 3, 6, 8, 11^2, 15, 19, & 26 : 3, 6, 8, 21, 23, & 44. & 27 : 6, 8, 12, 16, 19, \\
23, 24, 26, 33. & 25, 27. & 28 : 5, 8, 9, 19, 25, 38, & 23, 42. \\
29 : 5, 7, 10, 13, 14, 33, & 30 : 6, 8, 25^2, 30. & 44. & (12)
\end{align*}
1 : 2, 12, 14, 20, 30.  2 : 2, 11, 17, 20^2.  3 : 2, 11, 14, 20, 26, 29.  4 : 13, 22, 27, 28, 30.
5 : 2^2, 7, 18, 23, 32.  6 : 11, 12, 14, 20, 24.  7 : 2, 14, 20, 36.  35.
11 : 9, 14, 35, 37, 39.  12 : 2, 3, 11, 12^2, 19.  13 : 2, 3, 4, 12, 25, 35.  14 : 14, 20^2, 28, 33.
40  20, 21, 33^2, 34.  15 : 3, 15, 20, 30, 35.  16 : 5, 11, 14, 20.
17 : 4, 11, 16, 20, 25.  35^2, 38, 39.  18 : 2, 8, 12, 28, 30, 35.  19 : 2, 11, 14, 20.
35.  20 : 2, 10, 12, 14, 20.  21 : 2, 4, 20, 30, 35, 38.  22 : 2, 14^2, 31, 32, 35.
23 : 2, 3, 21, 26, 35.  24 : 9, 11, 12^2, 27, 28.  25 : 2, 14, 19, 20, 25.  26 : 2, 7, 11, 12, 23, 34.
27 : 9, 19, 21, 33, 35^2.  28 : 22, 23, 35^2.  29 : 4, 14, 20, 30, 31, 39.  30 : 1, 9, 12, 20, 22.  (13)
1 : 3, 21, 29.  2 : 10, 21, 24.  3 : 11, 12, 14, 21.  4 : 6, 12, 21, 30.  5 : 18, 21, 29.
6 : 8, 20, 21^2, 23.  7 : 7, 9, 11.  8 : 3, 8, 21, 28, 30.  9 : 16^2, 18.  10 : 10, 21, 25, 26.
11 : 4, 11, 25.  12 : 11, 25, 29.  13 : 5, 11, 25.  14 : 2, 16, 21, 24.  15 : 21, 24, 27.
16 : 11, 25^2.  17 : 1, 3, 14, 18, 21^2.  18 : 8, 9, 17, 21.  19 : 7, 14, 21, 25.  20 : 11, 24, 25, 30.
21 : 11, 15, 22.  22 : 3, 9, 13, 30.  23 : 19, 21^2.  24 : 8, 11, 20.  25 : 3^2, 7, 13, 14,
26 : 8, 11, 21.  27 : 15, 25, 29.  19, 24, 25, 30.  (14)
1 : 2, 5, 9, 12, 13.  2 : 13, 15, 18, 23.  3 : 6, 10, 12, 13.  4 : 8, 10, 13, 17.  5 : 7, 13, 15.
17, 23.  6 : 13, 15, 17.  7 : 6, 7, 11, 15.  8 : 4, 7, 13, 18.  9 : 7, 14, 15, 17.
10 : 7^2, 12, 21.  11 : 3, 7^2.  18.  12 : 7, 13, 15, 19.  13 : 7, 13, 18^2, 24.
14 : 7^2, 17, 20.  15 : 2, 7, 12.  16 : 7^2, 9, 13, 22.  17 : 9, 13^2, 15^2.  18 : 6, 7, 9, 13.
19 : 7^2, 12.  20 : 7, 11, 18, 22^2.  21 : 1, 7, 12, 15.  16, 17.
24.  (15)