A Rate-Compatible Family of Protograph-Based LDPC Codes Built by Expurgation and Lengthening

Sam Dolinar
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA, USA
e-mail: sam@shannon.jpl.nasa.gov

***Abstract*— We construct a protograph-based rate-compatible family of low-density parity-check (LDPC) codes that cover a very wide range of rates from 1/2 to 16/17, perform within about 0.5 dB of their capacity limits for all rates, and can be decoded conveniently and efficiently with a common hardware implementation. In contrast to alternative methods that create codes of different rates by puncturing, shortening, or expurgation, our method uses a combination of expurgation and lengthening (equivalent to an unusual extension) to produce lower-rate codes. Advantages compared to the alternative methods include roughly uniform utilization of common family decoder hardware for different rates, implementation with uniformly low maximum check node degrees despite high maximum rate, and a large fixed portion of the protograph that can be labeled with a fixed set of edge permutations for all rates.**

We apply this method to create a rate-compatible code family anchored by a particular code of (nominal) rate 7/8 and length $n = 8176$ designed by Kou, Lin and Fossorier [1] whose edge permutations are determined by Euclidean geometries (EG). All members of our family retain all of the EG-designed edges and circulant permutations of this anchor code, and this helps to avoid weak spots in the code graph that usually arise when edge permutations are assigned by greedy algorithms. There are also varying numbers of *auxiliary* and *ancillary* checks, variables, and edges, allowing realization of different rates, with fixed (nominal) dimension $k = 8176$ for all rates. Simulations show that all members of this family achieve steeply falling error rate curves without detectable error floors, at least to codeword error rates of about 10^{-6}.

I. Introduction

Our aim in this paper is to construct a nicely structured rate-compatible family of high-performance low-density parity-check (LDPC) codes. A rate-compatible family is a set of codes of different rates that can be decoded conveniently and efficiently using a common hardware implementation. All codes in the family should offer uniformly near-capacity performance. It is desirable that each code be expanded from a projected graph [2] or protograph [3] with a block-circulant (quasi-cyclic) structure for ease of decoder implementation.

In this paper we describe by example a method for constructing a rate-compatible family of protograph-based codes by a combination of expurgation and lengthening. Expurgation is accomplished by splitting check nodes in the protograph, and lengthening is accomplished by attaching dedicated accumulators to each new check.

[1]This work was funded by the IND Technology Program and performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

II. Code Family Construction by Expurgation and Lengthening

We start with an anchor code $\mathcal{C}_2 * Z$ of (nominal) rate 7/8 developed by Kou, Lin and Fossorier [1], and proposed as a standard for space applications by Goddard Space Flight Center [4]. This code is a purely regular (4,32) Gallager code; its asymptotic iterative decoding threshold on an additive white Gaussian noise (AWGN) channel is 3.35 dB, which is 0.51 dB from the capacity limit for rate-7/8 codes. The code $\mathcal{C}_2 * Z$ is constructed from a protograph $\mathcal{C}_2$ with 16 variable nodes, 2 check nodes, and 64 edges, illustrated in Fig. 1, where each edge is labeled by a $Z \times Z$ circulant permutation with $Z = 511$. The particular set of 64 circulants was chosen by the elegant Euclidean geometry (EG) method detailed in [1]. Unlike methods such as progressive edge growth (PEG) [5]

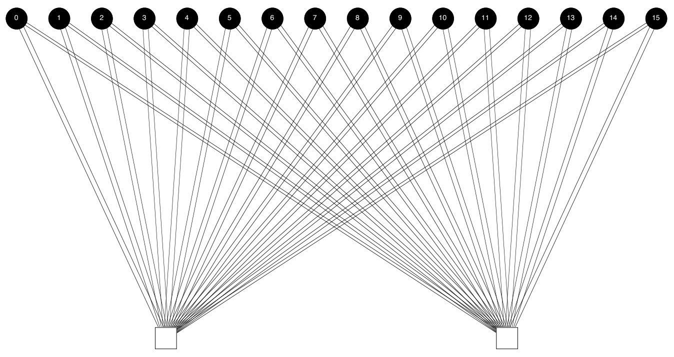

Fig. 1. The regular (4,32) anchor protograph $\mathcal{C}_2$.

or approximate cycle extrinsic message degree (ACE) [6] that select permutations greedily in a predetermined edge order, this expansion method avoids designing weak spots into the graph of the full-size code. The expanded code $\mathcal{C}_2 * Z$ has length $n = 16Z = 8176$ and nominal dimension $k = 14Z = 7154$; its actual dimension is $k = 7156$ due to two redundant checks. Note that in our notation $\mathcal{C}_2$ denotes the protograph and $\mathcal{C}_2 * Z$ the expanded code, whereas $\mathcal{C}_2$ denotes the full-size code in the notation of [4].

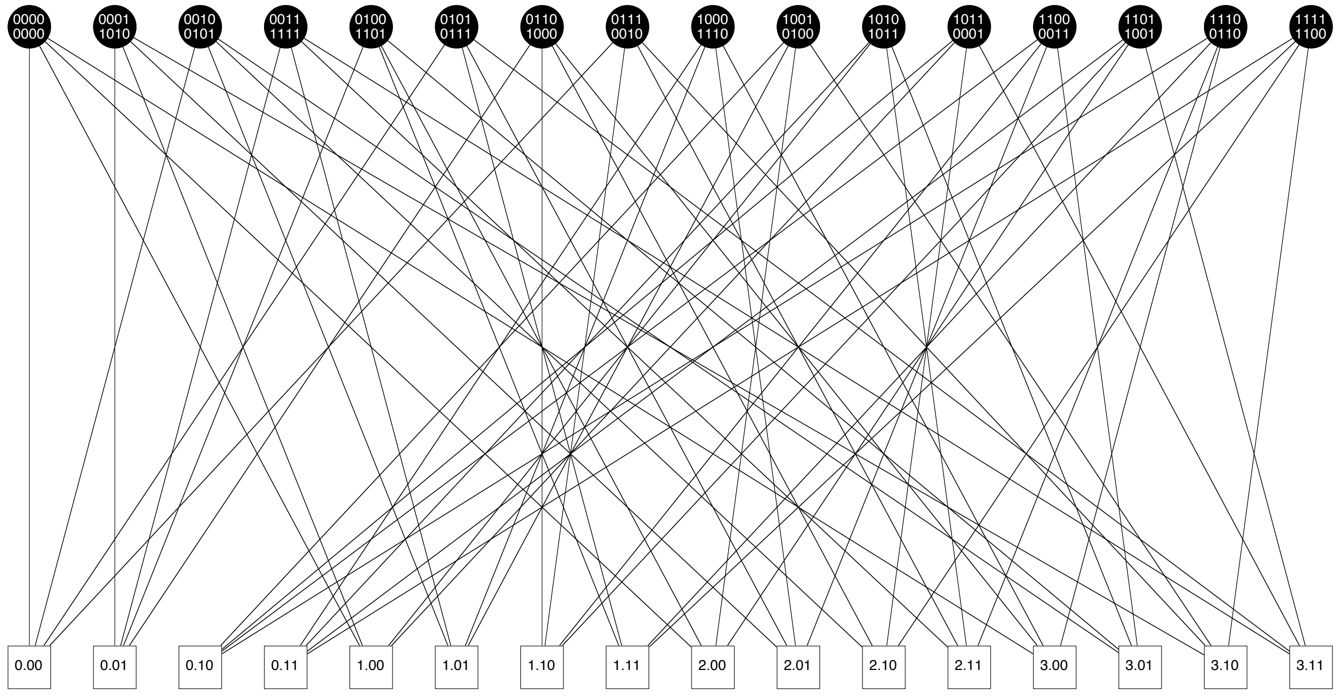

Fig. 2. The expurgated protograph $\mathcal{C}_{16}$ of (nominal) rate 0, illustrating check node splitting based on a $(4,2)$ quaternary code of distance 3.

The protograph $\mathcal{C}_2$ anchors a natural family of protographs of different rates obtained by splitting or merging check nodes. By merging the two check nodes in $\mathcal{C}_2$, we obtain a protograph $\mathcal{C}_1$ with (nominal) rate 15/16. By successively splitting the protograph's check nodes into four, then eight, then sixteen checks, we obtain additional family members $\mathcal{C}_4$, $\mathcal{C}_8$, $\mathcal{C}_{16}$, with (nominal) rates 3/4, 1/2, 0, respectively.

Our specific node-splitting rules for constructing the low-rate members of this family are designed to maximally and symmetrically spread each protograph's interconnections between its variable nodes and check nodes. First, $\mathcal{C}_4$ is obtained by splitting the two check nodes of $\mathcal{C}_2$ into pairs such that $\mathcal{C}_4$ has no parallel edges; this yields four check nodes for $\mathcal{C}_4$ numbered 0, 1, 2, 3. Each of the 16 variable nodes of $\mathcal{C}_4$ is connected by one edge to each of its four check nodes. Next, $\mathcal{C}_8$ is obtained from $\mathcal{C}_4$ by splitting each check node $c = 0, 1, 2, 3$ into a pair of check nodes $(c.0, c.1)$ and connecting the corresponding edge from variable node v to either $c.0$ or $c.1$ according to value of the c^{th} bit in the 4-bit binary representation of the numeric label of v. With this rule for splitting the check nodes, any two variable nodes in $\mathcal{C}_8$ share at most three neighboring check nodes. Finally, $\mathcal{C}_{16}$ is obtained from $\mathcal{C}_8$ by further splitting each check node $c.0$ or $c.1$ into a pair of nodes $(c.00, c.01)$ or $(c.10, c.11)$, respectively. The 16 variable nodes in $\mathcal{C}_{16}$ are labeled by two rows of 4 bits each, with each 2-bit column representing a code symbol of a $(4,2)$ quaternary code with distance 3. The first row is the same 4-bit label used to create $\mathcal{C}_8$ from $\mathcal{C}_4$. The second row determines the second level of node splitting used to create $\mathcal{C}_{16}$ from $\mathcal{C}_8$. With this rule for splitting the check nodes, any two variable nodes in $\mathcal{C}_{16}$ share at most one neighboring check node, because the underlying quaternary code defining the node splits has distance 3 and length 4. This implies, for example, that the $\mathcal{C}_{16}$ protograph is carefully constructed to possess no loops of length 4. Our application of these node-splitting rules is illustrated in Fig. 2 for the protograph $\mathcal{C}_{16}$.

Since all of these protographs retain the same 64 edges present in the anchor protograph $\mathcal{C}_2$, they can all be expanded to the same length $n = 16Z$ using the same set of 64 EG-designed $Z \times Z$ circulants. We denote the resulting family of expanded codes by $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_4, \mathcal{C}_8, \mathcal{C}_{16}\} * Z$.

Splitting check nodes in the protograph is equivalent to *expurgation* and produces a sequence of codes of successively lower rates with constant length n. This technique has been proposed, e.g., [7], as a favorable method for constructing rate-compatible code families. Others have constructed code families by *shortening* a high-rate code, e.g., [8], or by *puncturing* a low-rate code, e.g., [9], to obtain codes of successively lower or higher rates, respectively.

The protograph family $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_4, \mathcal{C}_8, \mathcal{C}_{16}\}$ constructed by expurgation achieves reasonable decoding thresholds at high rates, but very poor thresholds at low rates. The protographs in this family are regular $(4, d)$ Gallager codes with $d = 64, 32, 16, 8, 4$, respectively. The decoding threshold of 1.53 dB for the rate-1/2 protograph $\mathcal{C}_8$ is 1.34 dB worse than the capacity limit, and the protograph $\mathcal{C}_{16}$ corresponds to a meaningless code of (nominal) rate 0.

To improve threshold performance relative to rate-dependent capacity limits, we modify this family by attaching a dedicated accumulator to each check node in the protograph, obtaining *lengthened* protographs $\{\mathcal{C}_1^+, \mathcal{C}_2^+, \mathcal{C}_4^+, \mathcal{C}_8^+, \mathcal{C}_{16}^+\}$, of rates 16/17, 16/18, 16/20, 16/24, and 16/32, respectively. Each dedicated accumulator is represented in the protograph by an *ancillary variable node* connected to its corresponding check node by two edges, as described in the next section and illustrated in Fig. 3 on the next page.

This process of expurgation followed by lengthening keeps $k = 16Z$ constant while increasing n, for the codes $\{\mathcal{C}_1^+$, $\mathcal{C}_2^+$, $\mathcal{C}_4^+$, $\mathcal{C}_8^+$, $\mathcal{C}_{16}^+\} * Z$ expanded from these protographs. It has the same effect as *extension*, but it is extension in a non-straightforward way, because the new parity symbols sent to the channel are not simply computed from new check nodes inserted into the graph while keeping the rest of the graph unchanged.

III. Family Hardware Decoder Implementation

Fig. 3 on the next page depicts how the entire family of expurgated and lengthened codes, $\{\mathcal{C}_1^+, \mathcal{C}_2^+, \mathcal{C}_4^+, \mathcal{C}_8^+, \mathcal{C}_{16}^+\} * Z$, can be decoded with common decoder hardware. The 64 edges in the upper part of the graph are labeled with the 64 EG-designed $Z \times Z$ circulant permutations, identically for all codes in the family. All codes in the family use the same 16 variable nodes at the top of the graph and the same 4 edges emanating from each, as well as the same 16 numbered check nodes and the same 4 edges connecting each of these checks to the fixed set of 16 variables. The only variability from code to code is in the bottom part of the graph.

Besides the 16 numbered checks, Fig. 3 shows (in light red) an additional 15 *auxiliary check nodes* and 30 *auxiliary edges* attached to the auxiliary checks. Also shown (in light green) are 16 degree-2 *ancillary variable nodes* and 32 *ancillary edges* attached to the 16 numbered checks, representing the 16 dedicated accumulators in the protograph $\mathcal{C}_{16}^+$. The 30 auxiliary edges are all labeled with identity permutations. Without loss of generality, one of each pair of ancillary edges can also be labeled with the identity permutation, and the other permutation within each pair is a circulant selected by PEG. The 16 nontrivial circulants on the ancillary edges define how the 16 ancillary accumulators interact with the fixed part of the graph.

The auxiliary checks and edges, and the ancillary variables and edges, are activated or deactivated according to the schedule shown in Fig. 4 in order to achieve any of the desired rates. Deactivating an auxiliary or ancillary edge from an unused

Protograph	a	b	c	d	e	w	x	y	z
$\mathcal{C}_1^+$	+					+	+	+	+
$\mathcal{C}_2^+$	+	+					+	+	+
$\mathcal{C}_4^+$	+	+	+					+	+
$\mathcal{C}_8^+$	+	+	+	+					+
$\mathcal{C}_{16}^+$	+	+	+	+	+				

Fig. 4. Activation schedule for ancillary variable nodes and edges of types {a, b, c, d, e}, and auxiliary check nodes and edges of types {w, x, y, z}, as labeled in Fig. 3.

check or variable means setting its message to an infinitely reliable hard-decision 0 where it is connected to an active check node; this is equivalent to reducing the degree of its neighboring active check node by 1. For protograph $\mathcal{C}_1^+$, all of the auxiliary checks and edges are active, and all except one ancillary variable and its pair of edges are inactive. At the other extreme, for protograph $\mathcal{C}_{16}^+$, all of the auxiliary checks and edges are deactivated, while all of the ancillary variables and edges are active. For all rates, the activation/deactivation schedule is such that exactly 32 of the 62 total auxiliary and ancillary edges (as well as all 64 of the original fixed edges) are active.

Rate-compatible families generated by puncturing or shortening do not achieve such uniformity of hardware utilization across rates. Decoders for families created by puncturing are designed for the lowest-rate code, and successively greater numbers of edge messages are deactivated for higher rates. Decoders for families created by shortening are designed for the highest-rate code, and successively more variables and edges are deactivated as the rate is lowered. By contrast, our expurgated and lengthened family achieves relatively uniform hardware utilization over a very wide range of rates from 1/2 to 16/17.

In contrast with families created by shortening, our family $\{\mathcal{C}_1^+$, $\mathcal{C}_2^+$, $\mathcal{C}_4^+$, $\mathcal{C}_8^+$, $\mathcal{C}_{16}^+\}$ uses relatively uniformly low check-node degrees despite covering a very wide range of rates. This is possible because the cascade of auxiliary check nodes in Fig. 3 active for higher rates effectively reproduces the internal computations of a single high-degree check node in the usual graph for a high-rate code. Thus, our explicit introduction of auxiliary check nodes and auxiliary edges can be regarded as a method to make the internal check node computational complexity relatively uniform across rates.

A final advantage of our family compared to one obtained by shortening is that codes of a given dimension k are all obtained by using same-size circulants on the protographs of different rates. For families created by shortening, each shortened protograph has smaller dimension, and the circulant size Z for the corresponding expanded code must be increased in order to maintain constant dimension k.

IV. Further Improvement of Decoding Threshold

Our construction method has thus far produced a family of protographs of different rates that are all variants of a generalized family of semi-regular "repeat-4-and-accumulate" (R4A) codes. While the decoding thresholds achieved by this family are much better than those achieved by the completely regular $(4, d)$ Gallager codes, there is still plenty of room for further improvement.

We improve the $\{\mathcal{C}_1^+$, $\mathcal{C}_2^+$, $\mathcal{C}_4^+$, $\mathcal{C}_8^+$, $\mathcal{C}_{16}^+\}$ family of protographs by allowing one or more of their 16 original fixed variable nodes to be "pre-coded" in the style of accumulate-repeat-accumulate (ARA) codes [10]. For example, the protograph obtained by pre-coding of variable node 0 in protograph $\mathcal{C}_4^+$ is denoted by $\mathcal{C}_4^+[0]$. Here pre-coding means puncturing the pre-coded variable node in the protograph and attaching a new degree-3 check node with two edges connected to the

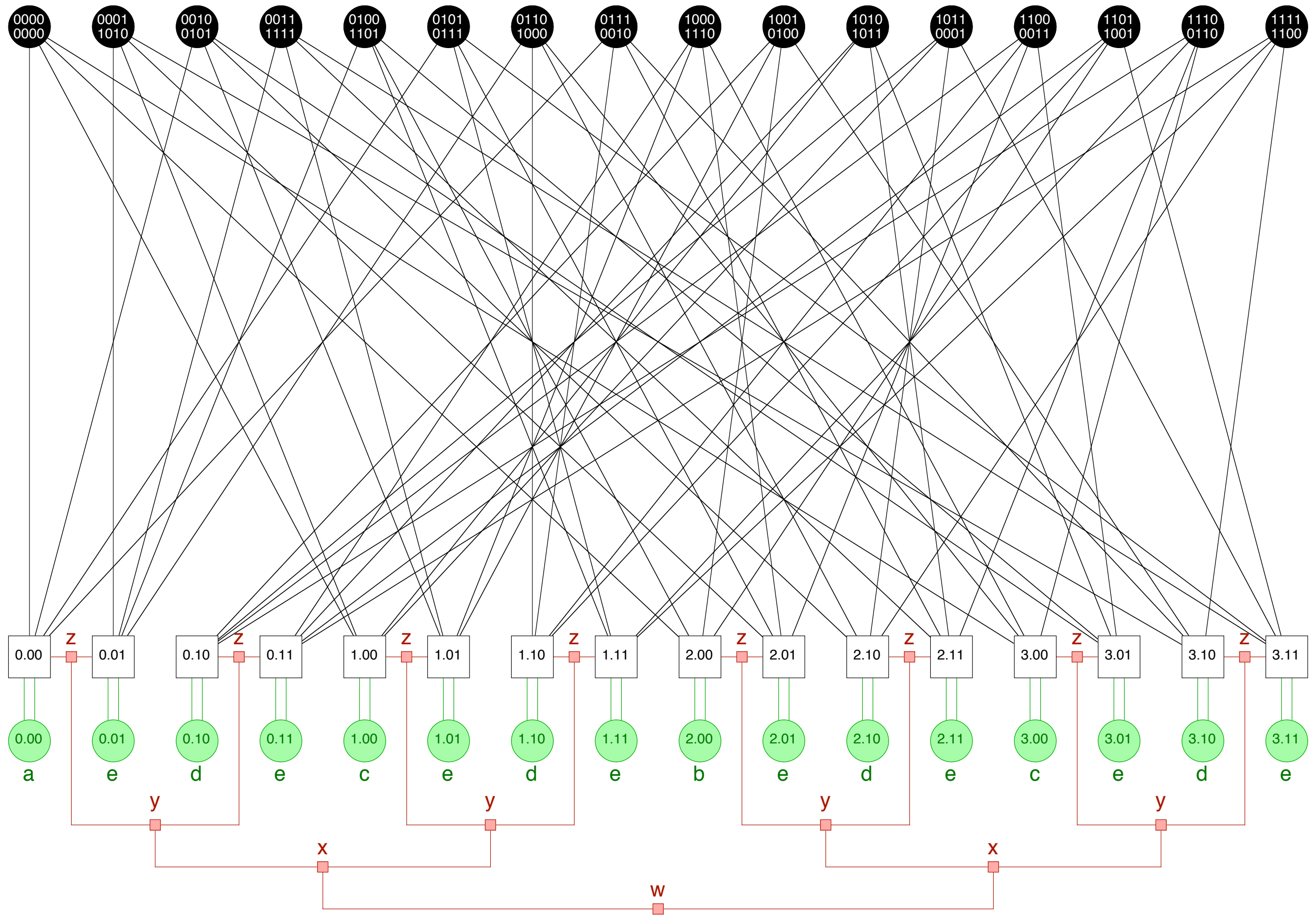

Fig. 3. Common family hardware decoder implementation for codes expanded from $\{\mathcal{C}_1^+, \mathcal{C}_2^+, \mathcal{C}_4^+, \mathcal{C}_8^+, \mathcal{C}_{16}^+\}$.

pre-coded variable and one edge connected to a new degree-1 variable sent to the channel. Fig. 5 shows the graph fragment that replaces a pre-coded punctured variable node; here the white circle denotes the original variable, now punctured, and the black circle denotes the new transmitted variable.

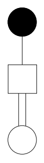

Fig. 5. The graph fragment replacing a punctured pre-coded variable node.

This type of resculpting of the protograph does not change k or n, but can improve the decoding threshold slightly. If more than one variable node is to be pre-coded, we require for simplicity that this same pre-coding rule be applied to each pre-coded variable separately. More complicated pre-codings of combinations of variable nodes are possible, but do not offer large further reductions in threshold.

No pre-coding of this type is possible for $\mathcal{C}_1^+$ or $\mathcal{C}_2^+$, because all of the check nodes would be disabled on the first iteration due to inputs from punctured pre-coded variables. Similarly, only one variable can be pre-coded in this way for $\mathcal{C}_4^+$. The maximum numbers of pre-codable variables are 5 and 9 for $\mathcal{C}_8^+$ and $\mathcal{C}_{16}^+$, respectively. Each new pre-coding of a variable typically lowers the decoding threshold by at least a few hundredths of a dB. However, our most robust code constructions do not use maximal pre-coding. Codes constructed from maximally pre-coded protographs usually require excessive iterations for convergence, and can produce less steep falloff in their error curves. Thus, we have opted for a set of *lightly pre-coded* protographs $\mathcal{C}_4^+[0]$, $\mathcal{C}_8^+[0,15]$, $\mathcal{C}_{16}^+[0,3,12,15]$, for which only 1, 2, 4 variables, respectively, are pre-coded. For light pre-coding, we require that all check nodes can compute at least one useful output on the first iteration despite the puncturing of the pre-coded variables. Violation of this condition seems to cause a substantial increase in the total number of iterations, not just slow convergence at the beginning of the decoding cycle.

The family decoder hardware for implementing the lightly pre-coded family will require 4 additional pairs of degree-3 ancillary checks and degree-1 ancillary variables, along with their corresponding 12 additional ancillary edges. The new ancillary checks, variables, and edges are simply activated or deactivated depending on whether the corresponding fixed variable nodes are to be pre-coded for the given rate.

Fig. 6 plots the iterative decoding thresholds achieved by the $\{\mathcal{C}_i\}$ family, the $\{\mathcal{C}_i^+\}$ family, the lightly pre-coded $\{\mathcal{C}_i^+[\text{light}]\}$ family, and the maximally pre-coded $\{\mathcal{C}_i^+[\max]\}$ family, relative to the corresponding rate-dependent capacity limits. We see that light pre-coding of the $\{\mathcal{C}_i^+\}$ family is sufficient to keep the decoding threshold within about 0.4 dB to 0.5 dB of the capacity limits for all rates in the family.

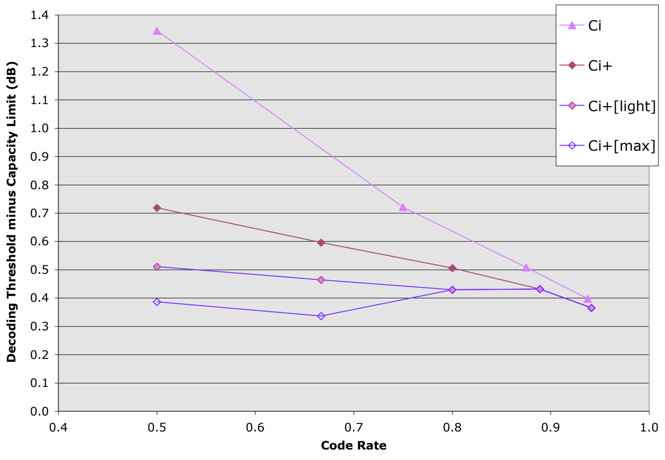

Fig. 6. Iterative decoding thresholds above the capacity limits for several code families.

V. Performance and Complexity Comparisons

Figs. 7 and 8 show error performance and decoding complexity of our codes of rates 1/2, 2/3, 4/5, with and without light pre-coding, expanded to $k = 8176$ using the 64 EG-designed circulants of [1] or [4] with $Z = 511$. Performance on an AWGN channel is shown as codeword error rate (WER) versus E_b/N_0 in Fig. 7. Complexity is shown in Fig. 8 as WER versus M_b, defined as the total number of edge messages per decoded bit computed in the decoding graph throughout all iterations. All of the performance curves in Fig. 7 show a consistently steep falloff in error rate and no error floor above WER = 10^{-6}, to the limits of detectability of our software simulation.

We see from Figs. 8 and 7 that light pre-coding costs just over 1 dB in iterative decoding complexity in return for only a small improvement of 0.1 dB to 0.2 dB in error performance at low WER. For heavier pre-coding the marginal return is even worse (not shown). In a practical coding system, the small reductions in iterative decoding threshold achievable by light or maximal pre-coding should be weighed against the significant increases in decoding complexity required to realize them.

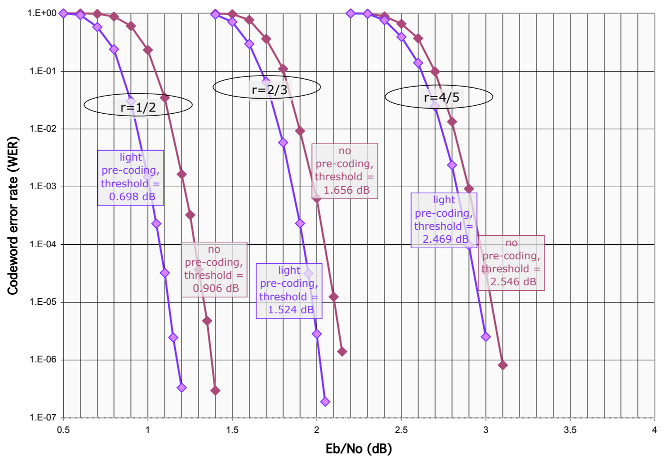

Fig. 7. Error performance of $\{\mathcal{C}_4^+, \mathcal{C}_8^+, \mathcal{C}_{16}^+\} * 511$, with and without light pre-coding.

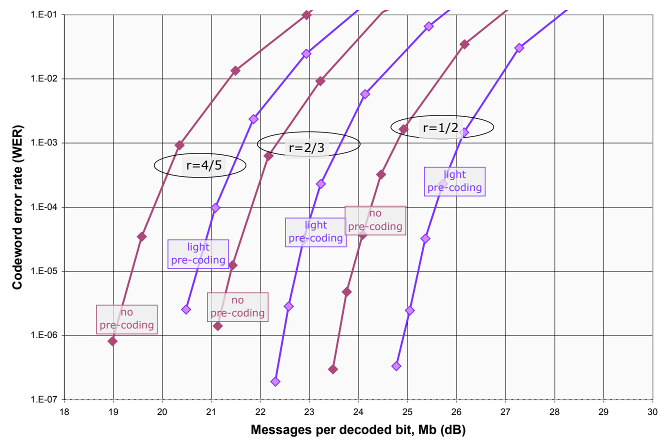

Fig. 8. Complexity of decoding $\{\mathcal{C}_4^+, \mathcal{C}_8^+, \mathcal{C}_{16}^+\} * 511$, with and without light pre-coding.

References

[1] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, pp. 2711–2736, Nov. 2001.

[2] T. Richardson and V. Novichkov, "Methods and apparatus for decoding LDPC codes." United States patent 6,633,856, Oct. 2003.

[3] J. Thorpe, "Low-Density Parity-Check (LDPC) Codes Constructed from Protographs," IPN Progress Report 42-154, JPL, Aug. 2003.

[4] W. Fong, S. Lin, G. Maki, and P.-S. Yeh, "Low Density Parity Check Codes: Bandwidth Efficient Channel Coding," in *NASA Earth Science Technology Conference 2003*, (College Park, Maryland), June 2003.

[5] D. M. Arnold, E. Eleftheriou, and X. Y. Hu, "Progressive edge-growth Tanner graphs," in *IEEE Global Telecommunications Conference 2001*, (San Antonio, Texas), pp. 995–1001, Nov. 2001.

[6] T. Tian, C. Jones, J. VIllasenor, and R. D. Wesel, "Characterization and selective avoidance of cycles in irregular LDPC code construction," *IEEE Transactions on Communications*, pp. 1242–1247, Aug. 2004.

[7] A. V. Casado, "Multiple Rate Low-Density Parity-Check Codes with Constant Blocklength." JPL Section 331 Seminar, Nov. 2004.

[8] T. Tian, C. Jones, and J. D. VIllasenor, "Rate-Compatible Low-Density Parity-Check Codes," in *IEEE International Symposium on Information Theory*, (Chicago, Illinois), June 2004.

[9] J. Ha and S. W. McLaughlin, "Analysis and design of punctured LDPCCs over Gaussian channel with erasures," in *IEEE International Symposium on Information Theory*, (Lausanne, Switzerland), June 2002.

[10] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate Repeat Accumulate Codes," in *IEEE International Symposium on Information Theory*, (Chicago, Illinois), June 2004.