Selective Avoidance of Cycles in Irregular LDPC Code Construction

Tao Tian, Chris Jones, John D. Villasenor, and Richard D. Wesel [*†‡]

December 20, 2003

Abstract

This work explains the effect of graph connectivity on error-floor performance of low-density parity-check (LDPC) codes under message passing decoding. A new metric called extrinsic message degree (EMD) measures cycle connectivity in bipartite graphs of LDPC codes. Using an easily computed estimate of EMD, we propose a Viterbi-like algorithm that selectively avoids small cycle clusters that are isolated from the rest of the graph. This algorithm is different from conventional girth conditioning by emphasizing the connectivity as well as the length of cycles. The algorithm yields codes with error floors that are orders of magnitude below those of random codes with very small degradation in capacity approaching capability.

Index Terms – irregular low-density parity-check codes, iterative decoding, message passing, graph cycles, extrinsic message degree.

1 Introduction

Low-density parity-check (LDPC) codes were proposed by Gallager in the early 1960s [1]. The structure of his codes (uniform column and row weight) led them to be called regular LDPC codes. Gallager tested codes with block lengths on the order of hundreds of bits.

[*]A subset of this work was presented at the 2003 IEEE International Conference on Communications.

[†]The authors are with the Electrical Engineering Department, University of California, Los Angeles, CA 90095, USA. e-mail: {ttian, christop, villa}@icsl.ucla.edu, wesel@ee.ucla.edu.

[‡]This work was supported in part by ONR grant N00014-01-C-0016 and NSF grant CCR-0209110.

However, these codes were too short for the sphere packing bound to approach Shannon capacity, and the computational resources for longer random codes were decades away from being broadly accessible.

MacKay [2] re-established interest in LDPC codes during the mid to late 1990s. Luby *et al.* [3] formally showed that properly constructed irregular LDPC codes can approach capacity more closely than regular ones. Richardson *et al.* developed the density evolution theory to analyze and synthesize the *degree distribution* in asymptotically large random bipartite graphs under a wide range of channel realizations.

Density evolution determines the performance threshold for infinitely long codes whose associated bipartite graphs are assumed to be tree-like. However, bipartite graphs of finite-length codes without singly connected nodes inevitably have cycles and thus are non-tree-like. If cycles exist, neighbors of a node are not independent in general, therefore Pearl's polytree algorithm [4] is inaccurate, so is the commonly practiced message passing decoding.

The tradeoff between threshold SNR and error floor BER has been observed for both turbo codes [5], [6] and LDPC codes [7]. At block size 10^4, rate 1/2 regular LDPC codes with variable node degree larger than two [2] enjoy very low error floors, however their threshold SNR is 0.6dB worse than their irregular counterpart [8]. In this paper, we repeated the irregular code construction method described in [8] and extended their simulation to a higher SNR region. In the relatively unconditioned case, an error floor was observed at BERs of slightly below 10^{-6}.

Conventional girth conditioning techniques increase the girth (shortest cycle length) of LDPC codes [9], [10]. We notice that not only the length but also the connectivity of cycles play important roles in error-floor performance. Since not all short cycles are equally harmful, selective treatment of more harmful cycles allows us to do cycle conditioning effectively. A minimum degree of connectivity is enforced among cycles shorter than a prescribed length. Some short cycles with good graph connectivity are allowed while longer cycles with poor graph connectivity are precluded. This technique lowers the error floors of irregular LDPC codes significantly while only slightly degrading the waterfall region performance.

Di *et al.* [11] described stopping sets in the context of erasures. Message passing decoding fails whenever all the variable nodes in a stopping set are erased. It is an open issue to find graph structures deterring message passing decoding in AWGN channels. As demonstrated by our simulations, we can reduce error floors in AWGN channels by generating codes from an expurgated ensemble that has good graph connectivity.

2 Cycle Clustering

The well known matrix and bipartite graph descriptions of a rate 1/3 $(9, 3)$ code are given in Fig. 1. This code will be used in examples throughout the paper. One column in the parity-check matrix corresponds to one variable in the bipartite graph. For convenience, we will use 'column' and 'variable' interchangeably in this paper. The parity-check matrix H is constructed such that its H_2 portion is invertible, which guarantees that H is full-rank. For systematic encoding, H_1 corresponds to information bits and H_2 corresponds to parity bits.

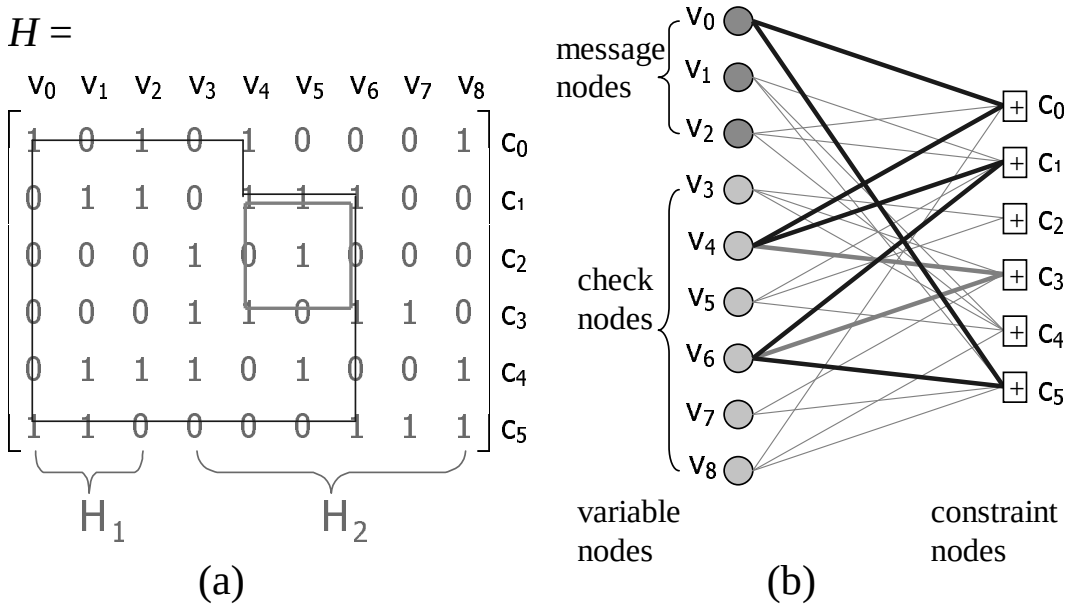


Figure 1: Matrix and graph description of a (9, 3) code. A stopping set $\{v_0, v_4, v_6\}$ and its cycles are illustrated.

Di *et al.* defined stopping sets as follows.

Definition 1 *A variable node set is called a* **stopping set** *if all its neighbors are connected to this set at least twice.*

Variable node set $\{v_0, v_4, v_6\}$ in Fig. 1 is a stopping set because all its neighbors c_0, c_1, c_3 and c_5 are connected to this set at least twice.

By default we assume LDPC codes without singly connected variables such as one generated with a degree distribution given by density evolution.

Lemma 1 *In a bipartite graph free of degree-1 variables, every stopping set contains cycles.*

Proof: A stopping set (variable nodes) and its neighbors (constraint nodes) form a non-tree-like sub-graph because a tree has singly connected leaf nodes.□

Stopping sets in general are comprised of multiple cycles. The only stopping sets formed by a single cycle are those that consist of all degree-2 variable nodes. A generic stopping sets is illustrated in Fig. 2. It is comprised of two types of variable nodes. Variable nodes of the first type form cycles with other variable nodes; variable nodes of the second type form binding structures that connect different cycles. It should be noted that both binding nodes and cycle nodes may have branches that lead to cycles containing variable nodes not in the current stopping set.

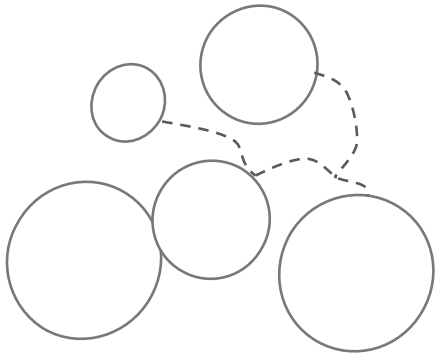


Figure 2: Structure of a stopping set.

For erasure channels, the effectiveness of message passing decoding depends primarily on how cycles are clustered to form stopping sets. Our proposed parity matrix design algorithm ensures that short cycles contain at least a given minimum number of 'extrinsic paths', leading to an increase in the minimum size of stopping sets.

Stopping sets entail codewords as a special case. Erasing all the non-zero positions of a binary codeword cannot be recovered under ML decoding as well as under message passing decoding. However, small stopping sets do not necessarily represent low distance events. Indeed, an ML decoder can successfully decode an erased stopping set if a full column-rank sub-matrix is formed by the columns indexed by the stopping set variables.

3 High-Degree Variables and Girth Conditioning

The vast existence of cycles challenges girth conditioning. Fig. 5 illustrates the support tree of variable node v_0 of Fig. 1. All the levels whose indices are odd numbers consist of

constraint nodes and all the levels whose indices are even numbers consist of variable nodes. A cycle occurs if two positions in the support tree represent the same node in the bipartite graph (*e.g.*, v_8 in level-3). To detect cycles of length up to $2d$ in the support tree of v_0, we need to expand its support tree d levels.

The number of nodes in the support tree grows exponentially with the number of levels expanded. To be short-cycle-free, all these nodes have to be different, so the longest cycle size we can avoid increases only logarithmically with block size (see [1]). Since the logarithm is a slowly increasing function, girth conditioning of a finite length LDPC code is severely limited by block length.

High-degree nodes are common with degree distributions produced by density evolution. They spawn many children in the support tree and pose a serious challenge to girth conditioning. This can be alleviated by using alternative degree distributions [9] [10] other than those predicted by density evolution. As a result, girth conditioning became easier to perform, however at the price of deviation from the asymptotically good density evolution distributions. The following arguments further discuss the easiness of cycle conditioning with respect to the degree of variable nodes.

Definition 2 *A variable node set is called a* **cycle-free set** *if no cycle exists among its constituent variables.*

Lemma 2 *A necessary and sufficient condition for a set of degree-2 variable nodes to be a cycle-free set is that this set is linearly independent.*

Proof: All sets that are not linearly independent contain codewords. Codewords are special stopping sets and stopping sets contain cycles (Lemma 1). For sufficiency, note that the constraint nodes taking part in a cycle among degree-2 nodes are each shared by exactly two variable nodes. Therefore the binary sum of columns (variables) taking place in the cycle is the all-zero vector and these columns are linearly dependent.□

Corollary 0.1 *A maximum of $n-k-1$ degree-2 columns of length $n-k$ may be linearly independent (cycle-free).*

Proof: Consider the $(n-k) \times (n-k-1)$ bi-diagonal matrix,

$$\left[\begin{array}{ccccccc} 1 & 0 & \cdots & & & 0 \\ 1 & 1 & & & & \\ 0 & 1 & \ddots & & & \\ \vdots & & \ddots & & & \vdots \\ & & & & 1 & 0 \\ & & & & 1 & 1 \\ 0 & \cdots & & & 0 & 1 \end{array}\right]_{n-k}, \tag{1}$$

$n-k-1$

This matrix forms a rank $n-k-1$ basis of degree-2 columns each with dimension $n-k$. Any possible degree-2 column of dimension $n-k$ can be formed via a linear combination of columns in the above basis.□

Theorem 1 *In an (n, k) code free of degree-1 variables, a cycle-free variable node set* $\{v_1, v_2, ..., v_s\}$ *must satisfy* $\sum_{i=1}^{s}(d_i - 1) \leq n-k-1$, *where* d_i *is the degree of* v_i.

Proof: A degree-d variable node whose constraints are $\{c_1, c_2, ..., c_d\}$ can be conceptually replaced by a cluster of $d-1$ degree-2 nodes whose constraints are $\{c_1, c_2\}$, $\{c_2, c_3\}$, ..., $\{c_{d-1}, c_d\}$ respectively. As an example, Fig. 3 shows how variable node v_1 in Fig. 1 may be replaced by a cluster of two degree-2 nodes.

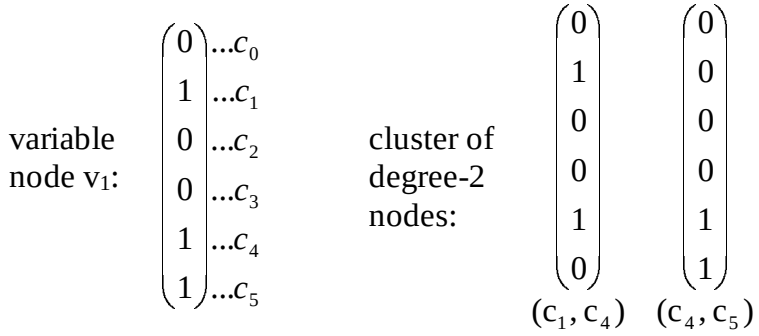


Figure 3: v_1 can be replaced by two degree-2 nodes.

Because the indices of the constraints in a cluster are ordered, any cycle involving the degree-d node is equivalent to a cycle involving some of the degree-2 nodes in the cluster replacing the original node. Fig. 4 shows an example. Replace every variable in the set with its equivalent cluster. According to Corollary 0.1, at most $n-k-1$ of these resulting degree-2 nodes can form a cycle-free set. Thus the theorem is proved.□

The inequality of Theorem 1 shows that lower degree variables allow a cycle-free set to contain more variables. Specifically, the equality is achieved with $n-k-1$ independent

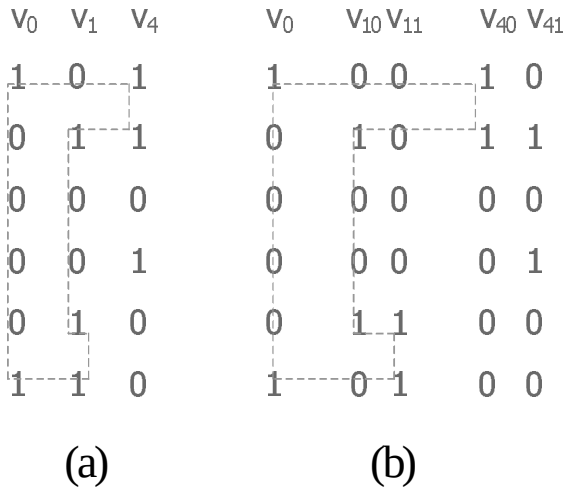


Figure 4: Replace v_1 by its cluster in a cycle.

degree-2 variable nodes. Higher degree nodes may be considered as the superposition of multiple degree-2 nodes. As a result, cycle conditioning on high degree nodes is more difficult than on low degree nodes. However, cycles involving higher degree nodes are actually less problematic because they have higher EMD, as will be explained in the next section.

4 Low-Degree Variables and EMD

Low-degree variable nodes widely exist in a density evolution optimized irregular LDPC codes. As will be shown in this section, low-degree variables are susceptible to channel noise because they contribute little to EMD and form isolated sub-graphs easily.

We define the notion of an edge expander as presented in [11].

Definition 3 *Let L be any subset of left nodes (variable nodes). Define $E(L)$ to be the number of edges connected to L and $N(L)$ to be the number of neighbors of L. An (α, γ) edge expander of an (n, k) code is a graph that has $N(L) > \gamma E(L)$ for all subsets with $E(L) \leq \alpha n$.*

Methods that realize regular graphs with good edge expansion properties were proposed by Margulis in [12]. However, a construction that can simultaneously satisfy a given edge expanding property as well as a given irregular degree distribution has yet to be proposed.

An efficient way to suppress small stopping sets is to improve the edge expanding properties of cycles in an irregular LDPC code. Specifically, an (α, 1/2) edge expander is free of

stopping sets with size below or equal to αn. This is because for any variable subset with size below or equal to αn, its neighbors (constraints) are connected to the subset less than twice on average, and therefore at least one neighbor is singly connected to this subset. Constraint nodes singly connected to a variable node set increase the expansion of the variable node set effectively. Based on this observation, we propose a parameter of variable node sets that we call the extrinsic message degree (EMD). This parameter works for AWGN channels because good edge expanders have been shown to perform well in various channel realizations.

Definition 4 *(***Extrinsic message degree***) An extrinsic constraint node of a variable node set is a constraint node that is singly connected to this set. The extrinsic message degree (EMD) of a variable node set is the number of extrinsic constraint nodes of this set.*

As was previously described, the high-SNR performance of an iteratively decoded LDPC code in erasure environment is limited by the size of the smallest stopping set. The EMD of a stopping set is zero. A set of variable nodes with large EMD will require additional concatenation of nodes to become a stopping set. We propose a conditioning algorithm that ensures all cycles less than a given length have an EMD greater than a given value. This technique statistically increases the smallest stopping set size.

The algorithm that follows ignores cycles whose length is above a threshold. In the context of EMD, this is justified for two reasons. First, large cycles necessarily contain many variable nodes. Second, they tend toward high EMD by virtue of a level of graph connectivity that statistically surpasses that of small cycles. We thus target the elimination of small cycles with low EMD.

Consider the EMD of a generic cycle. If there are no variable nodes in a cycle that share common constraint nodes outside of the cycle, then the EMD of this cycle is $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable in this cycle. Otherwise, the EMD is reduced through constraint node sharing. To provide a calculable EMD metric, we neglect constraint node sharing and define an approximate cycle EMD.

Definition 5 *(***Approximate cycle EMD (ACE)***) The ACE of a length $2d$ cycle is $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable in this cycle. We also say that the ACE of a degree-d variable node is $d - 2$ and the ACE of any constraint node is 0.*

ACE is an upper bound on EMD. The code conditioning algorithm to be proposed next is based on ACE instead of EMD. This approximation is reasonable since in this algorithm,

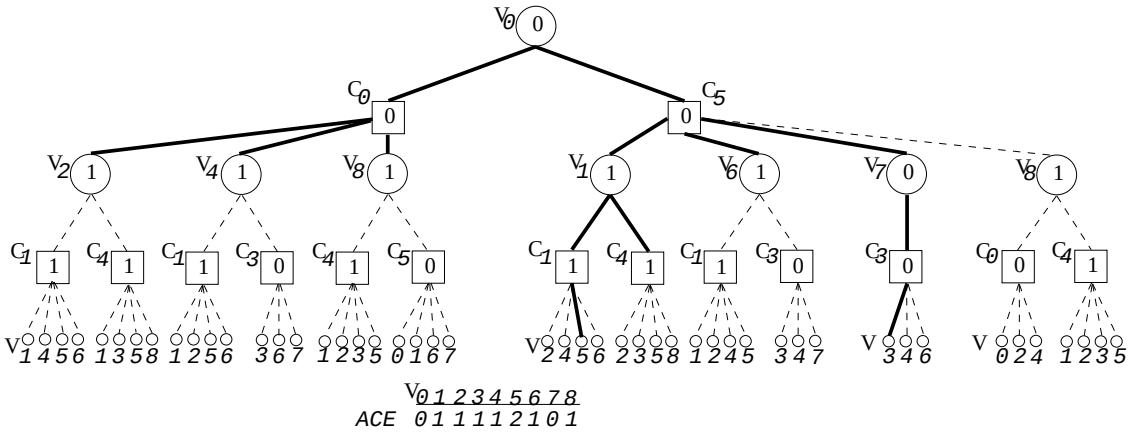


Figure 5: Illustration of an ACE search tree associated with v_0 in the example code of Fig.1. $\eta_{ACE} = 0$. Bold lines represent active paths. ACE values are indicated on the interior of circles (variables) or squares (constraints), except on the lowest level where they are instead described with a table.

since *all* cycles shorter than a given length (including those formed through constraint node sharing) will be required to meet the ACE criteria.

As demonstrated above, the ACE value of low-degree variables is low. Particularly, the ACE value of a degree-2 variable is zero. Correspondingly, low-degree variables form cycles with low EMD easily and low-EMD cycles have less connection with the rest of the graph. Poorly connected subgraphs are vulnerable to channel noise because they do not have sufficient impinging message flow to correct errors.

5 Constructing LDPC Codes with ACE algorithm

An LDPC code has property (d_{ACE}, η_{ACE}), if all the cycles whose length is $2d_{ACE}$ or less have ACE values of at least η_{ACE}.

In our codes information bits come before parity bits (see Fig. 1). We assign column nodes such that degree decreases monotonically (*i.e.*, $d_i \geq d_j$ if $i < j$). Because high degree nodes converge faster, this arrangement provides more protection to information bits than to parity bits. The algorithm is as follows:

for $(i = n-1;\ i \geq 0;\ i--)$
begin
redo:
  Randomly generate v_i according to *deg. distr.*;
  if $i \geq k$ (*i.e.*, v_i is a parity bit)
  begin
    Gaussian Elimination (GE) on H_2;
    if $v_i \in SPAN(v'_{i+1}, v'_{i+2}, ..., v'_{n-1})$
     goto *redo*;
    else
      $v'_i \leftarrow$ the residue of v_i after GE;
  end
  ACE detection for v_i;
  if $ACE < \eta_{ACE}$ for a cycle of length $2d_{ACE}$ or less
   goto *redo*;
end

The Gaussian elimination process ultimately guarantees that the H matrix has full rank by ensuring that the $n-k$ columns of H_2 be linearly independent. For degree-2 variable nodes, independence entails freedom from cycles so that all degree-2 parity check nodes will be cycle-free. A caveat is that if Gaussian elimination is used in conjunction with a degree distribution that yields more than $n-k-1$ degree-2 nodes, then at least one of the $n-k$ parity check variables should have odd number degree (this can be achieved by column swapping). This follows immediately from Corollary 0.1.

The ACE detection method can be equivalently depicted in two ways. The first one, based on support trees, is directly related to the graph structure. The second one, based on trellises, is oriented for algorithm implementation.

The tree depiction of ACE detection ($\eta_{ACE} = 0$) is given in Fig. 5. Here, variable and constraint node labels refer literally to those of the example code in Fig. 1 and the support tree that extends four levels below root node v_0 is portrayed. We define $p(\mu_t)$ to be the ACE of a path between root node v_0 and an arbitrary node μ_t (it can be either a variable node or a constraint node). Recall also that $ACE(\mu_t) = degree(\mu_t) - 2$ if μ_t is a variable, and $ACE(\mu_t) = 0$ if μ_t is a constraint.

ACE Detection of v_0

$p(\mu_t) \leftarrow \infty$ for all variables and constraints;
$p(v_0) \leftarrow ACE(v_0)$; Activate v_0 for level-0;
for $(l = 1; l \leq d_{ACE}; l++)$
begin
  for any active node w_s in level-$(l-1)$
  begin
    Find its children set $Ch(w_s)$;
    for every child $\mu_t \in Ch(w_s)$
    begin
      $p_{temp} \leftarrow p(w_s) + ACE(\mu_t)$;
      if $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) < \eta_{ACE}$[1]
        exit with failure;
      elseif $p_{temp} \geq p(\mu_t)$
        Deactivate μ_t in level-l with respect to current parent w_s;
      else
        $p(\mu_t) \leftarrow p_{temp}$;
    end
  end
end
exit with success;

To explain the above algorithm, we need to recognize that a node should propagate descendants (be active) only if the path leading to this node has the lowest ACE value that any path to this node has had thus far. Therefore linear cost is achieved instead of an exponential cost. Initially all the path ACEs can be set to ∞ (which means 'unvisited'). Note that cycles occur when a node is revisited or simultaneously visited. A *cycle ACE* equals the sum of the previously lowest path ACE to a node and the current path ACE to the node minus the doubly counted root and child ACE. When a cycle is formed by connecting two distinct paths from v_0 to μ_t we have cycle $ACE = p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)$, where p_{temp} and $p(\mu_t)$ are the ACEs of the two paths from v_0 to μ_t. Handling multiple

[1]Note that this is the ACE of a cycle involving μ_t if $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) < \infty$.

simultaneous arrivals to the same node is a trivial extension where ACE minimization is performed sequentially across all arrivals.

In the example shown in Fig. 5, bold lines at each level describe the current set of *active paths*. In this example 'ties' are assigned the path whose parent has the lowest index. For instance the path (v_0-c_5-v_1-c_1) with $ACE = 1$ survives while, (v_0-c_5-v_6-c_1), (v_0-c_0-v_2-c_1), (v_0-c_0-v_4-c_1) each also having $ACE = 1$, perish. For an example of pruning occurring due to cycle detection on differing levels of the tree, observe that the path (v_0-c_0-v_8-c_5) with $ACE = 1$ does not survive since c_5 was visited at Level-1 and was accordingly assigned $ACE = 0$.

Fig. 6 provides a trellis depiction of the previous discussion (with two more stages added). A trellis instead of a full support tree is adequate for ACE detection because the ACE minimization is performed sequentially and only the minimum ACE needs to be stored. Again, a path ACE is stored for every variable node and every constraint node. An active path is a path that connects the root variable node and any other node with the lowest ACE value up to the current step. Active paths are marked by solid lines in Fig. 6. An *active node* is a node that connects to an active path at the current step.

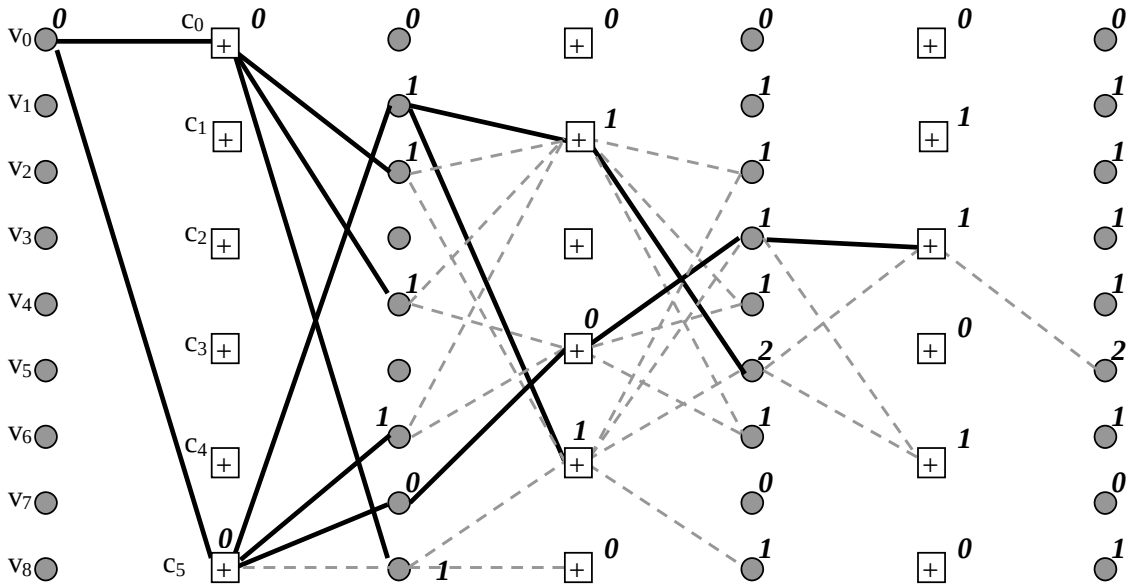


Figure 6: The Viterbi-like ACE algorithm. $\eta_{ACE} = 0$

Viterbi tree pruning yields a complexity at each root that is upper bounded by $d_{ACE} \times n \times (d-1)$ where d is the highest degree of any node in the graph, because the support tree is expanded d_{ACE} levels, for each level we have to consider at most n nodes, and every node

has at most $d-1$ children. As shown in Fig. 6, relatively few nodes at a level are active, thus the actual computational burden is reasonable, even for block size on the order of 10^5 bits. The storage space needed is on the order of $n+(n-k)$ because only the current trellis level has to be saved. To further improve run-time, any active node with path ACE $p(\mu_t) \geq \eta_{ACE}$ can be deactivated because all the paths stemming from this node have an ACE value of at least η_{ACE}. As a special application of this argument, the user may generate columns for all variable nodes with degree $\eta_{ACE}+2$ or higher without checking the ACE (since all descendants will have a path ACE of at least η_{ACE}).

Note that cycle detection can be implicitly performed in the above algorithm. Unvisited variables are initialized to $p(\mu_t) = \infty$. When a variable is first visited, $p(\mu_t)$ is assigned a finite value. Upon subsequent visits to the same variable, ill-conditioned cycle detection is alarmed if the condition $(p_{temp} + p(\mu_t) - ACE(v_0) - ACE(\mu_t)) \geq \eta_{ACE}$ is not satisfied.

6 Simulation results

We used the ACE algorithm to construct (10000, 5000) codes that have the irregular degree distribution given in [8] with maximum variable node degree $d_v = 20$. The encoded bits were sent through a binary-input AWGN channel. For each corrupted codeword, a maximum of 200 iterations were performed. Each simulation was stopped when 80 block errors were detected. The BER results and in one case the word (block) error rate (WER) results are plotted in Fig. 7.

A (d_{ACE}, η_{ACE}) code in Fig. 7 means that in this code, all cycles of length up to $2d_{ACE}$ have ACE of at least η_{ACE}. Higher η_{ACE} values ensure better properties for the conditioned cycles and higher d_{ACE} values ensure that conditioning occurs for longer cycles. There is a tradeoff between d_{ACE} and η_{ACE}: increasing one parameter inevitably makes the other parameter more difficult to implement. For $d_{ACE} = 13, 10, 9, 7$, and 6, the highest η_{ACE} values that we could achieve were $2, 3, 4, 5$, and 7 respectively. If all degree-2 variables are parity bits (which is possible if there are fewer than $n-k$ of them), then the full-rankness of the parity matrix guarantees that no cycles will exist between degree-2 variables (see Lemma 2). In this case we can construct (∞, 1) codes since all cycles of any length must include nodes of at least degree-3 (and hence have ACE of at least 1). Such codes have the lowest level of conditioning that we consider and we use codes constructed in this manner to contrast the performance of stronger (d_{ACE}, η_{ACE}) conditioning levels.

As a benchmark, Richardson and Urbanke's 10^4-bit code [8] (referred to here as the RU code) is included in Fig. 7(a). This RU code was constructed with the constraint that "all the degree-two nodes were made loop-free". Unfortunately, simulation results below BER 10^{-6} are unavailable for the RU code. The (∞, 1) code shown in our paper has a similar level of conditioning as the RU code and exhibit comparable performance. The (∞, 1) code ensures linear independence among parity bits which include all degree-two nodes and some degree-three nodes. The error-floor BER of the (∞, 1) code is around 10^{-6}. Pure length-4 cycle removal improves BER by half an order in magnitude over the (∞, 1) code at highest SNR, and adversely affects convergence. However, proper selection of d_{ACE} and η_{ACE} suppresses error floors significantly more. For example, the ACE conditioning code (9, 4) achieves approximately BER = 10^{-9}, three orders of magnitude below the (∞, 1) code.

Fig. 7(b) compares several ACE parameter sets with or without explicit length-4 cycle removal. Note that the lowest error floor was achieved at $d_{ACE} = 9$ and $\eta_{ACE} = 4$ with no explicit removal of short cycles. As explained before, traditional girth conditioning treats all short cycles equally, thus making the removal of longer but still harmful cycles difficult. On the contrary, the ACE algorithm effectively removes low-ACE cycles, while leaving shorter cycles intact, if they have a high ACE. By doing this, participation of high degree variables in cycles is encouraged.

We observe that there is a small penalty in the capacity-approaching capability of our low-error-floor codes. With more conditioning at this block size, the low-SNR performance of the code is slightly degraded, possibly due to a decrease in the randomness of the code structure. This tradeoff between error floor and low-SNR performance is a well-known characteristic of iteratively decoded codes ([5][6][7]). Scheme (9, 4) is approximately 0.07dB away from the RU code at low SNR. However, even with this mild penalty these codes remain superior to regular codes in terms of their capacity-approaching performance. For example, we tested MacKay's (9972, 3, 6) regular LDPC code described in [7]. Although no error floors were detected for this code, it achieves BER $\approx 10^{-5}$ at $SNR \approx 1.7$dB, more than 0.6dB worse than our (9, 4) code. Thus the combination of density evolution optimized degree distributions and ACE construction achieves good performance over a wide SNR operating range.

7 Conclusion

We have discussed the effect of graph connectivity on error floors and have introduced the ACE algorithm to construct irregular LDPC codes that have a specific ACE property for short cycles while maintaining the density-evolution prescribed degree distribution. Our simulation results show an improvement in the error floor region of irregular LDPC codes by several orders of magnitude in BER over unconditioned codes.

This paper assumes that asymptotically optimal degree distributions provide a good starting point for designing codes with short block lengths. Graphs with lower maximum left degree can achieve higher girth conditioning numbers than graphs with higher maximum left degree. The absence of high degree nodes, however, leads to threshold degradation. In contrast to this, Ryan [13] advocate elimination of *low* degree nodes from the distribution. The irregular LDPC code in [10] also picked a distribution free of degree-2 variables. Though not explicitly described in [13] and [10], removal of low degree nodes from a distribution will raise overall graph EMD since low degree nodes cannot substantially increase the EMD of any given cycle. According to density evolution, either rate or threshold is adversely affected by removing low-degree nodes, since a small fraction of low degree nodes can significantly raise code rate, while only marginally degrading threshold.

8 Acknowledgments

The authors would like to acknowledge the suggestions from many people, particularly Pascal Vontobel, Ksenija Lakovic, Aditya Ramamoorthy and Mike Smith.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[3] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585–598, Feb. 2001.

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

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909–926, May 1998.

[6] C. Fragouli and R. D. Wesel, "Bit vs. symbol interleaving for parallel concatenated trellis coded modulation," in *Proc. IEEE Global Telecommun. Conf.,* San Antonio, TX, Nov. 2001, vol. 2, pp. 931–935.

[7] D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," in *Proc. 36th Allerton Conf. Commun., Control and Comput.,* Sept. 1998.

[8] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638–656, Feb. 2001.

[9] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proc. IEEE Int. Conf. Commun.,* Helsinki, Finland, June 2001.

[10] D. M. Arnold, E. Eleftheriou, and X. Y. Hu, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Global Telecommun. Conf.,* San Antonio, TX, Nov. 2001, vol. 2, pp. 995–1001.

[11] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, June 2002.

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

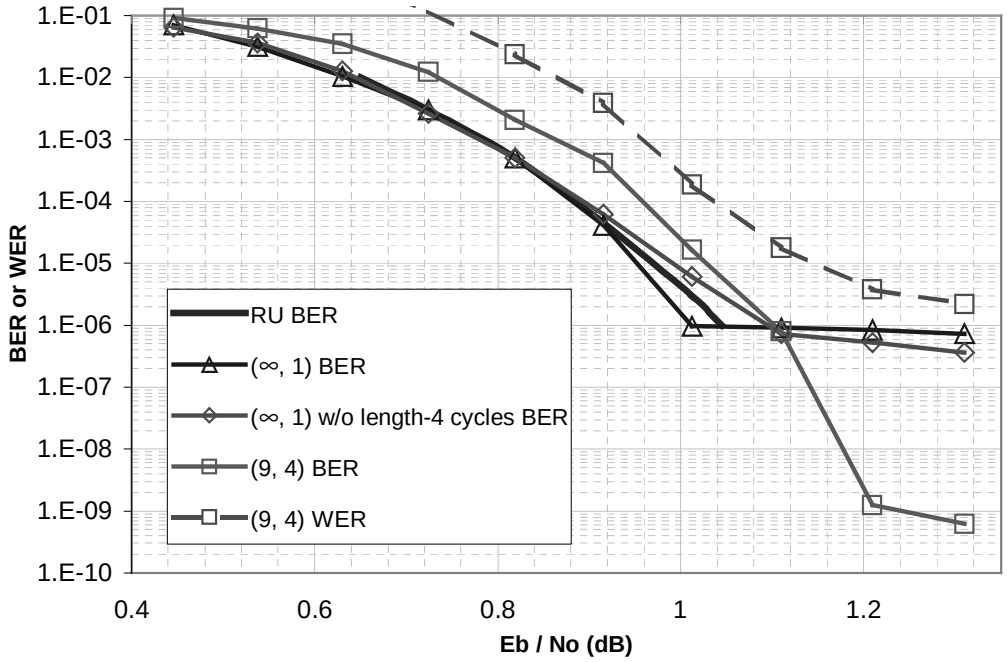
[13] M. Yang and W. E. Ryan, "Lowering the error-rate floors of moderate-length high-rate irregular LDPC codes," to appear in *IEEE Int. Symp. Infor. Theory,* Yokohama, Japan, June 2003.

[14] J. Yedidia, E. Sudderth, and J. Bouchaud, "Projection algebra analysis of error-correcting codes," in *Proc. 39th Allerton Conf. Commun., Control and Comput.,* Oct. 2001, pp. 662–671.

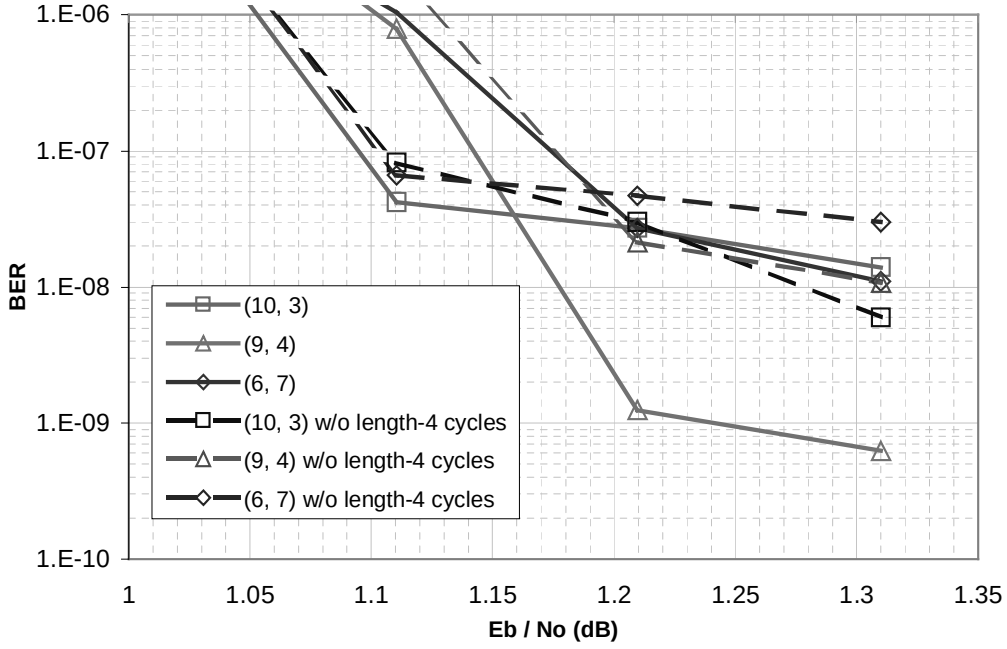
[15] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.

[16] S. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inform. Theory*, vol. 47, pp. 657–670, Feb. 2001.

[17] M. Fossorier, "Iterative reliability-based decoding of low-density parity-check codes," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 908–917, May 2001.



(a)



(b)

Figure 7: Results for (10000, 5000) codes.
The BPSK capacity bound at $R = 0.5$ is 0.188dB.