

Construction of Irregular LDPC Codes with Low Error Floors

Tao Tian, Chris Jones, John D. Villasenor, and Richard D. Wesel
 Electrical Engineering Department, University of California, Los Angeles, CA, 90095, USA
 {ttian, christop, villa}@icsl.ucla.edu, wesel@ee.ucla.edu

Abstract— This work explains the relationship between cycles, stopping sets, and dependent columns of the parity check matrix of low-density parity-check (LDPC) codes. Furthermore, it discusses how these structures limit LDPC code performance under belief propagation decoding. A new metric called extrinsic message degree (EMD) measures cycle connectivity in bipartite graphs. Using an easily computed estimate of EMD, we propose a Viterbi-like algorithm that selectively avoids cycles and increases stopping set size. This algorithm yields codes with error floors that are orders of magnitude below those of girth-conditioned codes.

I. INTRODUCTION

LOW-DENSITY parity-check (LDPC) codes have generated much interest recently. Richardson *et al.* [1] created the density evolution algorithm to analyze the *degree distribution* in asymptotically large random bipartite graphs whose associated bipartite graphs are assumed to follow a tree-like structure. However, bipartite graphs representing finite-length codes without singly connected nodes inevitably have cycles and thus are non-tree-like. Cycles in bipartite graphs compromise the optimality of the commonly practiced belief propagation decoding. If cycles exist, neighbors of a node are not conditionally independent in general, therefore graph separation [2] is inaccurate and so is belief propagation decoding.

Randomly realized finite-length irregular LDPC codes with block sizes on the order of 10^4 [1] approach their density evolution *threshold* closely (within 0.8dB) at rate 1/2, outperforming their regular counterparts [3] by about 0.6dB. In this paper, we repeated the irregular code construction method described in [1] and extended their simulation to a higher SNR region. In the relatively unconditioned codes, an error floor was observed at BERs of slightly below 10^{-6} . In contrast, regular codes usually enjoy very low error floors. MacKay *et al.* [4] first reported the tradeoff between the threshold SNR and the error floor BER for irregular LDPC codes versus regular LDPC codes. A similar tradeoff has been found for turbo codes ([5]).

The error floor of an LDPC code under maximum likelihood (ML) decoding depends on the d_{min} of the code and the multiplicity of d_{min} error events. However, directly designing an LDPC code for large d_{min} is computationally prohibitive since ensuring that every set of d_{min} columns of an $(n-k) \times n$ parity check matrix is linearly independent has a computational complexity of $\binom{n}{d_{min}}$.

As a result, the common approach has been to indirectly improve d_{min} through code conditioning techniques such as the removal of short cycles (girth conditioning [6], [7]). However, not all short cycles are equally harmful. Stan-

dard girth conditioning severely constrains code structure by removing all cycles shorter than a specified length even though many of these can do little harm, as will be explained. Under belief propagation decoding, small stopping sets [8] in random codes lead to high error floors. However, these can be alleviated by generating codes from an expurgated ensemble that has large stopping sets. This paper uses a technique that addresses only cycles that are likely contributors to small stopping sets. In this paper, we show that such a technique lowers the error floors of irregular LDPC codes significantly while only slightly degrading the performance in the waterfall region.

II. CYCLES, STOPPING SETS, LINEARLY DEPENDENT SETS AND EDGE-EXPANDING SETS

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 the H_2 portion of it 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.

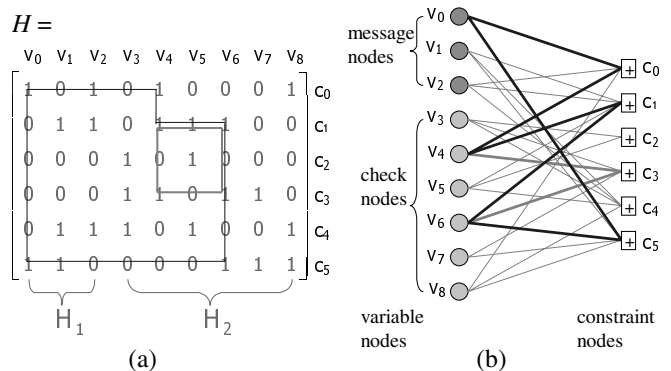


Fig. 1. Matrix and graph description of a (9, 3) code.

Definition 1: (Cycle) A cycle of length $2d$ is a set of d variable nodes and d constraint nodes connected by edges such that a path exists that travels through every node in the set and connects each node to itself without traversing an edge twice.

Definition 2: (C_d Cycle set) A set of variable nodes in a bipartite graph is a C_d set if (1) it has d elements, and (2) one or more cycles are formed between this set and its neighboring constraint set. A set of d variable nodes does

This work was supported in part by the Office of Naval Research.

not form a C_d set only if no cycles exist between these variables and their constraint neighbors.

Note that the maximum cycle length that is possible in a C_d is $2d$. Fig. 1 shows a length-6 cycle ($v_0 - c_0 - v_4 - c_1 - v_6 - c_5 - v_0$) and a length-4 cycle ($v_4 - c_1 - v_6 - c_3 - v_4$). Variable node set $\{v_0, v_4, v_6\}$ is a C_3 set. Variable node set $\{v_4, v_5, v_6\}$ is also a C_3 set although v_5 is not contained in the length-4 cycle.

Definition 3: (S_d Stopping set [8]) A variable node set is called an S_d set if it has d elements and all its neighbors are connected to it at least twice.

Variable node set $\{v_0, v_4, v_6\}$ in Fig. 1 is an S_3 set because all its neighbors c_0, c_1, c_3 and c_5 are connected to this set at least twice.

Di, *et al.* [8] showed that in a binary erasure channel (BEC), the residual set of erasures after belief propagation decoding is exactly equal to the maximum size stopping set that is a subset of the originally erased code symbols. The next theorem shows that stopping sets always contain cycles. The effectiveness of belief propagation decoding on graphs with cycles depends primarily on how cycles are clustered to form stopping sets.

Lemma 1: In a bipartite graph without singly connected variable nodes (such as one generated with a degree distribution given by density evolution), every stopping set contains cycles.

Proof: A stopping set (variable nodes) and its neighbors (constraint nodes) form a bipartite graph where one can always leave a node on a different edge than used to enter that node. Traversing the resulting bipartite graph in this way indefinitely, one eventually visits a node twice, thus forming a cycle. \square

Theorem 1: In a bipartite graph without singly connected variable nodes, stopping sets in general are comprised of multiple cycles. The only stopping set formed by a single cycle is one that consists of all degree-2 variable nodes.

Proof: Obviously a cycle that consists of all degree-2 variable nodes is a stopping set. To prove the theorem, we only need to show that if a cycle contains variable nodes of degree-3 or more, any stopping sets including this cycle are comprised of linked cycles. Fig. 2(a) shows a cycle of arbitrary length $2d$ (here $2d = 8$ for demonstration). Assume that one variable node in this cycle v_2 has degree 3 or higher, v_2 must bring in all its neighboring constraint nodes to form a stopping set. Let c_1 be an extrinsic neighbor of v_2 . By the definition of a stopping set, c_1 must be connected to variable nodes in the stopping set at least twice. Therefore if c_1 is not connected to v_1 , or v_3 , or v_4 , the stopping set must contain at least one more variable node (for instance v_5 in Fig. 2(a)). The ‘chaining’ of constraints and variables on to v_5 may occur across many nodes. However, to form a stopping set, eventually a new loop must be closed that connects the newest constraint in the chain to a variable on the chain or in the original cycle. \square

This argument produces the general view of stopping sets and cycles is given in Fig. 2(b). Two types of variable nodes comprise a stopping set. Variable nodes of the first type form cycles with other variable nodes; variable nodes of the second type form chaining structures that connect different cycles. Our proposed algorithm will construct par-

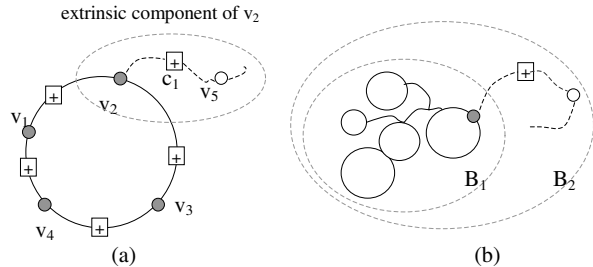


Fig. 2. (a) Extrinsic message (b) Expanding of a graph

ity matrices such that short cycles contain at least a given minimum number of ‘extrinsic paths’. This will ensure an increase in the average size of stopping sets.

Definition 4: (L_d Linearly dependent set) A variable node set is called an L_d set if it is comprised of exactly d elements whose columns are linearly dependent but any subset of these columns is linearly independent.

Variable node set $\{v_0, v_4, v_6\}$ in Fig. 1 is an L_3 set. A code with minimum distance d_{min} has at least one $L_{d_{min}}$ set but no L_d sets where $d < d_{min}$. The next theorem gives the relationship between L_d and S_d sets.

Theorem 2: A set of variable nodes (columns of H) that form an L_d set must also form an S_d set.

Proof: The binary sum of all columns corresponding to the variable nodes in L_d is the all-zero vector. Thus any neighbor (constraint node) of an L_d set is shared by the variable nodes in the set an even number of times, which means at least twice. \square

Theorem 2 implies that preventing small stopping sets also prevents small d_{min} . If a code has d_{min} , it must have an $S_{d_{min}}$ stopping set. Thus, avoiding all stopping sets S_d for $d \leq t$ ensures $d_{min} > t$.

The converse of Theorem 2 is untrue. 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 of the parity check matrix that are indexed by the stopping set variables. However, an erased stopping set can never be overcome by an iterative decoder. Fig. 3 summarizes the relationship between C_d , S_d , and L_d .

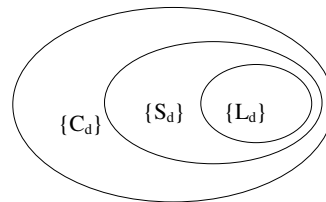


Fig. 3. Venn diagram showing relationship of C_d , S_d and L_d .

The role of stopping sets is easily translated to non-erasure scenarios where variables with poor observation reliability are analogous to erasures. Therefore, an obvious direction to take in order to generate codes well-suited to iterative decoding is to increase the minimum stopping set size. However, one might argue that simple girth conditioning accomplishes this. Since every stopping set contains cycles, removing all small cycles will certainly remove all

small stopping sets.

The problem with traditional girth conditioning is that there are so many cycles. Fig. 4 illustrates a cycle in 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.

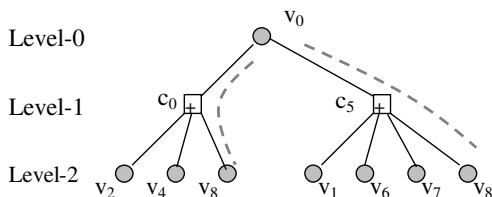


Fig. 4. Traditional girth conditioning removes too many cycles.

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. Since the logarithm is a slowly increasing function, girth conditioning of a finite length LDPC code is severely limited by block length.

Girth conditioning is especially problematic when there are high-degree nodes, as is common with degree distributions produced by density evolution. Recent girth conditioning techniques usually bypassed high degree nodes. For example, the highest variable degree in [6] was 3. As a result, girth conditioning was easier to perform, however, the capacity-approaching capability was sacrificed.

III. EMD AND LDPC CODE DESIGN

While our goal is to ensure that all stopping sets have at least some minimum number of variable nodes, our algorithm does not explicitly remove small stopping sets. Theorem 1 shows that stopping sets are comprised of linked cycles. An efficient way to suppress small stopping sets is to ensure cycles have enough neighbors that provide useful message flows. Our algorithm achieves this by focusing on a parameter of variable node sets that we call the extrinsic message degree (EMD).

Definition 5: (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 variable node set.

The EMD of a stopping set is zero. A set of variable nodes with large EMD will require additional ‘closure’ 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 to be described ignores large cycles. 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 and claim (via the arguments of previous sections) that these structures are significant contributors to errors at high SNR.

Now we 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 (the cycle contains no subcycles), 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 the approximate EMD of a cycle.

Definition 6: (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.

This approximation is reasonable since in the proposed algorithm, all cycles shorter than a given length will be required to meet the ACE criteria. An LDPC code has property (d_{ACE}, η) , if all the cycles whose length is $2d_{ACE}$ or less have ACE values of at least η .

We assign column nodes such that $d_i \geq d_j$ if $i < j$. Because high degree nodes converge faster, this arrangement provides more protection to information bits in H_1 than to parity bits in H_2 (see Fig. 1). The algorithm is as follows:

```

for ( $i = n - 1; i \geq 0; i --$ )
begin
  redo:
    Generate  $v_i$ ;
    if  $i \geq k$  (i.e.,  $v_i$  is a parity bit)
      begin
        Gaussian Elimination (GE) on  $H_2$ ;
        if  $v_i \in \text{SPAN}(v'_j)$  where  $i + 1 \leq j \leq n - 1$ 
          goto redo;
        else
           $v'_i \leftarrow$  the residue of  $v_i$  after GE;
        end
      ACE detection for  $v_i$ ;
      if  $\text{ACE} < \eta$  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 will 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.

Now we introduce two equivalent depictions of the ACE detection method. 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 ($\eta = 0$) 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_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

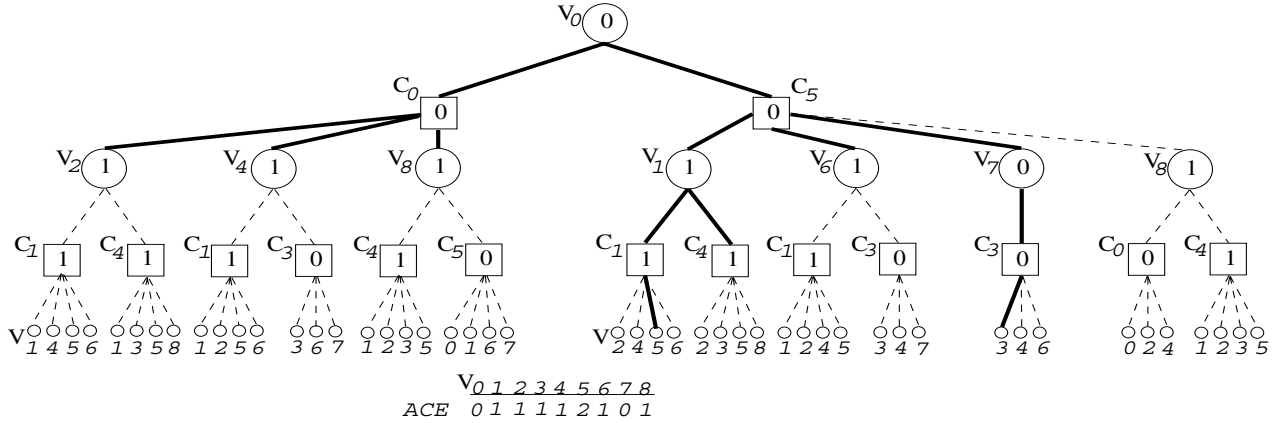


Fig. 5. Illustration of an ACE search tree associated with v_0 in the example code of Fig.1. Bold lines represent survivor paths. ACE values are indicated on the interior of circles (variables) or squares (constraints), except on the lowest level where they are described with a table.

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  $v_0$  (tree depiction)
 $p_t \leftarrow \infty$  for all variables and constraints;
 $p_0 \leftarrow ACE(v_0)$ ; Activate  $v_0$  for level-0;
for ( $l = 1$ ;  $l \leq d_{ACE}$ ;  $l++$ )
begin
  Expand all the active nodes in level- $(l-1)$  to level- $l$ ;
  for every node  $\mu_t$  in level- $l$ 
  begin
     $p_{temp} \leftarrow p(PARENT(\mu_t)) + ACE(\mu_t)$ ;
    if  $p_{temp} + p_t - ACE(v_0) - ACE(\mu_t) < \eta$ 
      exit with failure;
    elseif  $p_{temp} \geq p_t$ 
      Deactivate  $\mu_t$  in level- $l$ ;
    else
       $p_t \leftarrow p_{temp}$ ;
    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, is simultaneously visited, or both. 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. Handling multiple 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 $(v_0-c_0-v_8-c_5) =$

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 stages augmented). A trellis instead of a full support tree is enough 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.

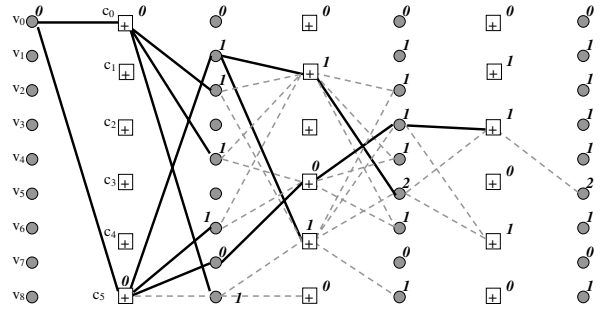


Fig. 6. The Viterbi-like ACE algorithm. $\eta = 0$

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, not all the nodes at a step are active, thus the actual computation burden is reasonable, even for block size on the order of 10^5 bits. To improve run-time, simply generate columns for all variable nodes with degree $\eta + 2$ or higher. This follows since all paths in their support trees have ACE values of at least $ACE_{root} \geq (\eta + 2) - 2 = \eta$.

IV. SIMULATION RESULTS AND DATA ANALYSIS

We used the ACE algorithm to construct (10000, 5000) codes that have an irregular degree distribution with a

maximum variable node degree of 20 (given in [1]). The encoded bits were sent through a binary-input additive white Gaussian noise channel. For each corrupted codeword, a maximum of 200 iterations were performed. The BER and word error rate (WER) results were plotted in Fig. 7.

A (d_{cyc}, d_{ACE}, η) code in Fig. 7 means this code is free of length $2d_{cyc}$ cycles, and all cycles up to length $2d_{ACE}$ have ACE of at least η . For the first parameter, two values were tested: $d_{cyc} = 1$ and 2, which represent no cycle removal and length-4 cycle removal respectively. The value of d_{ACE} corresponding to $\eta = 1$ is not specified because in this case all the cycles (of any length) could be forced to have at least one exit.

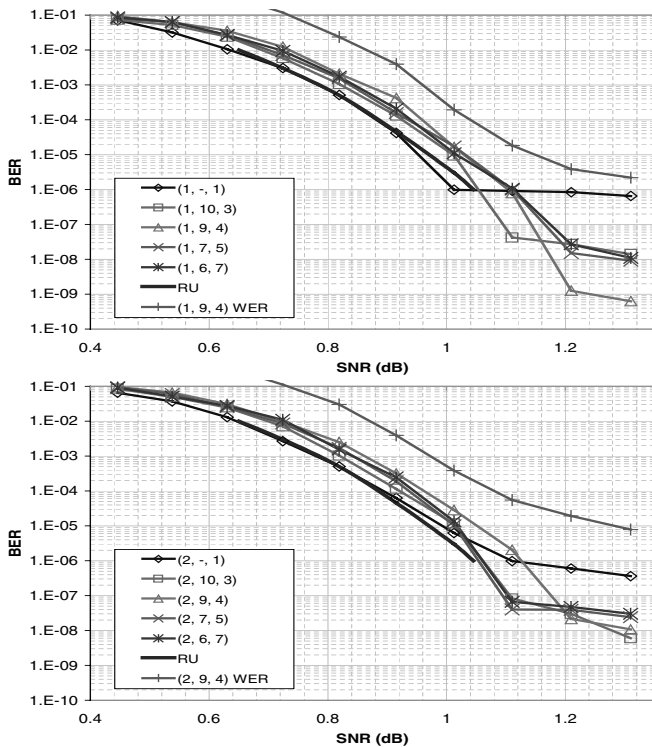


Fig. 7. Simulation results for (10000, 5000) codes. The BPSK capacity bound at $R = 0.5$ is 0.188dB.

There is a tradeoff between η and d_{ACE} . For both $d_{cyc} = 1$ and $d_{cyc} = 2$, the optimal choice is approximately $d_{ACE} \approx 9$ and $\eta \approx 4$. Increasing d_{cyc} from 1 to 2 has some effect in lowering the error floor only if η is small enough ($\eta \leq 3$).

As a benchmark, Richardson and Urbanke's 10^4 -bit code [1] (referred to here as the RU code) is included in Fig. 7. Our two codes $(1, -, 1)$ and $(2, -, 1)$ have a similar level of conditioning as the RU code and in turn exhibit comparable performance. Comparing scheme $(2, -, 1)$ (girth conditioning) with scheme $(1, -, 1)$ (no conditioning except for full-rankness), we see an error-floor BER improvement of at most half an order in magnitude. However, proper selection of d_{ACE} and η significantly suppresses error floors (the error floor of parameter construction $(1, 9, 4)$ is approximately $BER = 10^{-9}$).

It should be noted that there is a small penalty in the capacity-approaching capability of low-error-floor codes.

Scheme $(1, 9, 4)$ 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. Our $(1, 9, 4)$ code achieves $BER \approx 10^{-5}$ at a SNR level more than 0.6dB lower than MacKay's (9972, 3, 6) regular LDPC code described in [4]. Thus the combination of density evolution optimized degree distributions and ACE construction achieves good performance in both high and low SNR operating ranges.

Since the RU codes are asymptotically good for long blocks, at block sizes around 1000, we choose Mao's (1268, 456) code described in [6] as a benchmark. Fig. 8 compares the performance of the ACE-conditioned (1264, 456) code with that of Mao's (1268, 456) code. The degree distribution of the proposed code was generated by density evolution with $d_v = 14$.

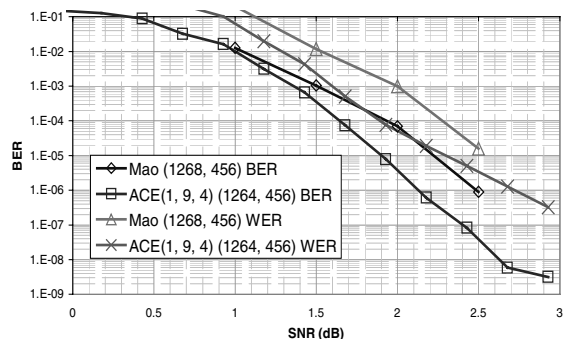


Fig. 8. The BPSK capacity bounds at 0.36 is -0.394dB.

V. CONCLUSION

We have discussed the relationship between several graph structures that affect error floors and introduced the ACE algorithm to construct irregular LDPC codes that have pre-designed ACE for short cycles. Our simulation results show an improvement in the error floor region of irregular LDPC codes by several orders of magnitude in BER.

REFERENCES

- [1] 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.
- [2] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Francisco, CA: Morgan Kaufmann, 1988.
- [3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.
- [4] 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.
- [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] 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.
- [7] 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.
- [8] 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.