Correspondence

Construction of Irregular LDPC Codes by Quasi-Cyclic Extension

Jinghu Chen, *Member, IEEE*, R. Michael Tanner, *Fellow, IEEE*, Juntan Zhang, and Marc P. C. Fossorier, *Fellow, IEEE*

***Abstract*—In this correspondence, we propose an approach to construct irregular low-density parity-check (LDPC) codes based on quasi-cyclic extension. When decoded iteratively, the constructed irregular LDPC codes exhibit a relatively low error floor in the high signal-to-noise ratio (SNR) region and are subject to relatively few undetected errors. The LDPC codes constructed based on the proposed scheme remain efficiently encodable.**

***Index Terms*—Low-density parity-check (LDPC) codes, quasi-cyclic extension, recursive code construction.**

I. Introduction

Irregular low-density parity-check (LDPC) codes [1], [2] have potentials to outperform regular ones in waterfall regions. However, good irregular LDPC codes are more difficult to construct than regular LDPC codes. An irregular LDPC code whose parity-check matrix (or Tanner graph) is randomly generated typically exhibits an error floor, even if its degree profile allows density evolution (DE) [3], [4] to predict a very good asymptotic threshold. This becomes prominent for LDPC codes of short to medium lengths. Moreover, degree profiles optimized with DE do not necessarily generate good LDPC codes of short to medium lengths.

The performance of an LDPC code depends on both the structure of its Tanner graph and its minimum distance (or low-weight profile). An LDPC code with a good minimum distance may not outperform another one with worse minimum distance but better graph structure, because the belief propagation (BP) decoding algorithm is suboptimum and graph dependent [5]. Therefore, most works on LDPC code design have looked for graphs suitable for iterative decoding, for example, by optimizing the girth [6] or other parameters such as *extrinsic message degree* (EMD) [7]. Although irregular LDPC codes designed to achieve good girth or EMD properties can achieve good weight distribution to some extent, they may still have low-weight codewords. The presence of a large number of low-weight codewords is critical to both the error floor and probability of undetected errors.

Manuscript received June 24, 2005. This work was supported by the National Science Foundation under Grants CCR-02-31099 and CCR-04-30576.

J. Chen was with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 USA. He is now with Qualcomm Inc., San Diego, CA 92121 USA (e-mail: jinghuc@qualcomm.com).

R. M. Tanner is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 USA (e-mail: rmtanner@uic.edu).

J. Zhang was with the Department of Electrical Engineering, University of Hawaii at Manoa, Honolulu, HI 96822 USA. He ia now with Availink Inc., Germantown, MD 20874 USA (e-mail: juntan@spectra.eng.hawaii.edu).

M. P. C. Fossorier is with the Department of Electrical Engineering, University of Hawaii at Manoa, Honolulu, HI 96822 USA (e-mail: marc@aravis.eng.hawaii.edu).

Communicated by T. J. Richardson, Associate Editor for Coding Theory.

Color versions of Figures 1–6 of this correspondence are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2007.892805

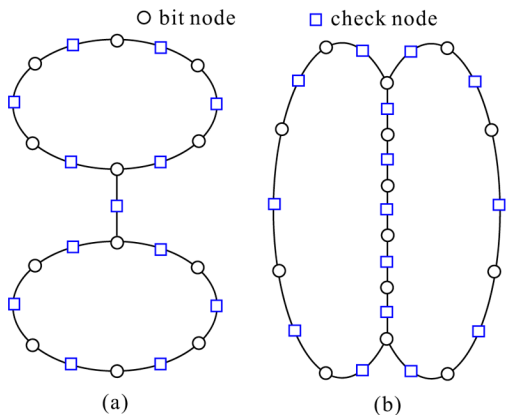

Fig. 1. Subgraphs that lead to low-weight codewords.

In this correspondence, we construct LDPC codes by taking both weight distribution and graph property into account. In particular, we eliminate low-weight codewords consisting of degree-2 bit nodes and at most two degree-3 bit nodes. Our design is based on the quasi-cyclic extension of [8], proposed to design repeat–accumulate (RA) codes with good minimum distance. Simulation results show that irregular codes designed with our approach have good weight distribution in that few undetected errors and low error floors are observed.

II. Limit of Code Constructions Based on Graph Properties

Most techniques on constructing irregular LDPC codes focus on searching for code graphs with good properties. However, for irregular LDPC codes with abundant bits of degree 2 and 3, large girth does not necessarily imply good distance property. As an example, suppose that there is a cycle constituted of only degree-2 bit nodes. By flipping all the degree-2 bits in that cycle, all the checks are still satisfied. Hence, there exists a codeword with weight equal to the number of degree-2 nodes in that cycle. Therefore, it has been suggested to eliminate all cycles containing exclusively degree-2 bits from the construction [2], [9]. Unfortunately, even if there is no cycle of degree-2 bits only, it is still possible to have low-weight codewords with a small number of degree-3 bits. Some examples are shown in Fig. 1. In Fig. 1(a) or (b), a number of degree-2 bits are linked by two bit nodes of degree 3. If all the bits in the subgraph are flipped, all checks still hold. Therefore, the corresponding LDPC code has a codeword of weight equal to the number of bit nodes in the subgraph. We can see that the total number of bit nodes for either Fig. 1(a) or (b) may be small, even though the lengths of the cycles *per se* in the subgraph might be acceptably large.

In [6], the girth of an LDPC code is optimized with a procedure called progressive edge-growth (PEG). However, as discussed earlier, a good girth property does not necessarily guarantee a good weight distribution. In [7], a parameter called ACE is defined for a cycle as $\sum_i (d_i - 2)$, where d_i is the degree of the ith variable node in the cycle. An LDPC code has property $(d_{\mathrm{ACE}}, \eta_{\mathrm{ACE}})$ if all the cycles of length $2d_{\mathrm{ACE}}$ or less have an ACE value of at least η_{ACE}. Methods to construct LDPC codes with good ACE properties have been proposed in [7]. Unfortunately, an LDPC code whose Tanner graph has good ACE property may still contain low-weight codewords. For example, the subgraph shown in Fig. 1(b) may exist in the Tanner graph of an

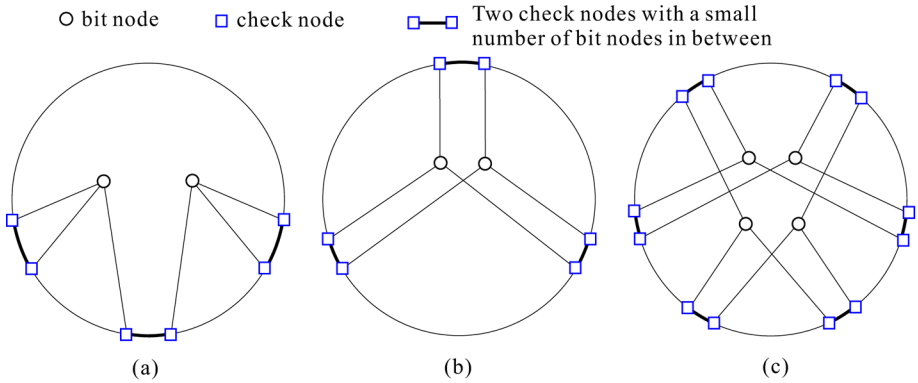

Fig. 2. Relations of two degree-3 nodes that lead to low-weight codewords.

LDPC code with $(d_{\mathrm{ACE}}, \eta_{\mathrm{ACE}}) = (9, 4)$, since the three cycles in Fig. 1(b) have all length 20.

III. Proposed Approach to Construct Irregular LDPC Codes

Suppose we intend to construct an LDPC code of length N and dimension K. The Tanner graph of the code has N_i bit nodes of degrees i, with $i = 2, 3, \ldots, d_{v,\max}$ and $N = \sum_{i=2}^{d_{v,\max}} N_i$. We assume that the number of degree-2 bit nodes is less than or equal to the number of check nodes, i.e., $N_2 \leq N - K$. The first step in our code construction is to alternately connect degree-2 bit nodes and check nodes to form a chain. Then, we introduce "tail-biting" in the chain to obtain a loop. Thereafter, we connect bit nodes of degree-3 or higher to check nodes on and off the loop.

A. Constraints Added to Connections of Bit Nodes

The connections of bit nodes of degree three or higher to check nodes on the tail-bited loop determine the property of the code. We take the following into consideration in connecting these higher degree bits.

1) Any two check nodes on the tail-bited loop connected to a bit node of degree 3 or higher are widely separated with at least d_1 degree-2 nodes in between. By this constraint, we can avoid not only some short cycles that are harmful to iterative decoding, but also some possible low-weight codewords of the form shown in Fig. 2(a). Note that Fig. 1(a) is a special case of Fig. 2(a).
2) Any two degree-3 bit nodes are widely separated. More specifically, consider the two degree-3 bit nodes connected to check nodes indexed by $\boldsymbol{c} = (c_1, c_2, c_3)$ and $\boldsymbol{c}' = (c_1', c_2', c_3')$. Then it is required that $\sum_{i=1}^{3} |c_i - c_i'| \geq d_3$ for any permutation of $\boldsymbol{c}$ and $\boldsymbol{c}'$, where $|c_i - c_i'| = \min(c_i - c_i', c_i' - c_i) \bmod N_2$ and d_3 is a design parameter. If this constraint is not satisfied, then the code has a codeword of weight less than $d_3 + 2$ since if we flip the two degree-3 bits and all degree-2 bits between check nodes c_i and c_i' ($i = 1, 2$, and 3) of a codeword, all check nodes still hold. This constraint eliminate subgraphs depicted in Fig. 2(b), which is essentially the same case as in Fig. 1(b).
3) In addition, we require $\sum_{i=1}^{2} |c_i - c_i'| \geq d_2$ for any permutation of $\boldsymbol{c}$ and $\boldsymbol{c}'$, where $d_2 < d_3$ is also a design parameter. This constraint is intended to improve the length of cycles consisting of degree-2 and degree-3 bits, in order to make the graph suitable for BP decoding.

However, to check whether all the degree-3 bits satisfy pairwise the previous conditions for chosen parameters d_2 and d_3 entails very large complexity. This complexity is of the order of $O({N_3}^2)$, where N_3 is the number of degree-3 bits. Fortunately, the introduction of quasi-cyclic extension in the following section simplifies this check process.

Our constraint on bit nodes of degree 3 is intended to eliminate low-weight codewords with two degree-3 bits and several degree-2 bits in their graph representation. On the other hand, there may still exist some bad configurations like that of Fig. 2(c), in which four degree-3 bits participate. Our hope is that such cases occur with a low probability. Consequently, although we cannot as yet guarantee a bound on the minimum distance, the weight distribution of the codes we have tested have been good when all three previous constraints are satisfied. Thus far we have made no effort to optimize the length of cycles that have bit nodes of degrees higher than three. Without additional tests, there may exist cycles of length four formed by bit nodes of high degrees.

B. Recursive Constructions With Quasi-Cyclic Extension

Quasi-cyclic extension permits the construction of a long graph-based code from a shorter one. In [8], quasi-cyclic extension was introduced to construct RA codes that can outperform randomly constructed RA codes in the error-floor region. RA codes can be considered as a special family of irregular LDPC codes, whose check nodes have a degree of three. For an RA code, each check node is connected to one and only one information bit node, while for an irregular LDPC code based on a loop, as above, each check node is connected to multiple bit nodes of degree three or higher. This makes the design of connections harder for our LDPC codes than for RA codes.

To construct a long LDPC code satisfying the constraints presented in Section III-A, we first generate a small graph, and then use quasi-cyclic extension to recursively generate a larger graph. The quasi-cyclic property of the graph allows us to readily estimate whether the graph is good or not, as well as facilitates the encoder and decoder designs in hardware.

The code construction process can be quickly described by a small example. Suppose we want to construct an $(N, K) = (288, 144)$ code, with $N_2 = 144$, $N_3 = 108$, and $N_8 = 36$. We start by constructing a small graph $\mathcal{G}'$, which has $N_2' = 12$ degree-2 bit nodes and 12 check nodes, forming a tail-bited loop as shown in Fig. 3. This graph also has $N_3' = 9$ degree-3 bit nodes and $N_8' = 3$ degree-8 bit nodes. The degree-3 and – 8 bit nodes are randomly connected to check nodes in $\mathcal{G}'$ such that every check node has a degree of either 6 or 7. After $\mathcal{G}'$ has been obtained, we associate to each edge e' on the graph an offset value $f(e')$. Then a larger graph $\mathcal{G}$ is constructed by repeating each node $v' \in \mathcal{G}' n$ times, $n = 12$, to obtain a group of nodes $\boldsymbol{v}, \boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. We call n the *extension factor*. Each edge $e' = (v', u')$ between two nodes $v' \in \mathcal{G}'$ and $u' \in \mathcal{G}'$, is also repeated n times to form a group of edges $\boldsymbol{e} = (e_1, e_2, \ldots, e_n)$. Then each edge $e_i, 1 \leq i \leq n$ is connected to one node in group $\boldsymbol{v}$ and one node in $\boldsymbol{u}$

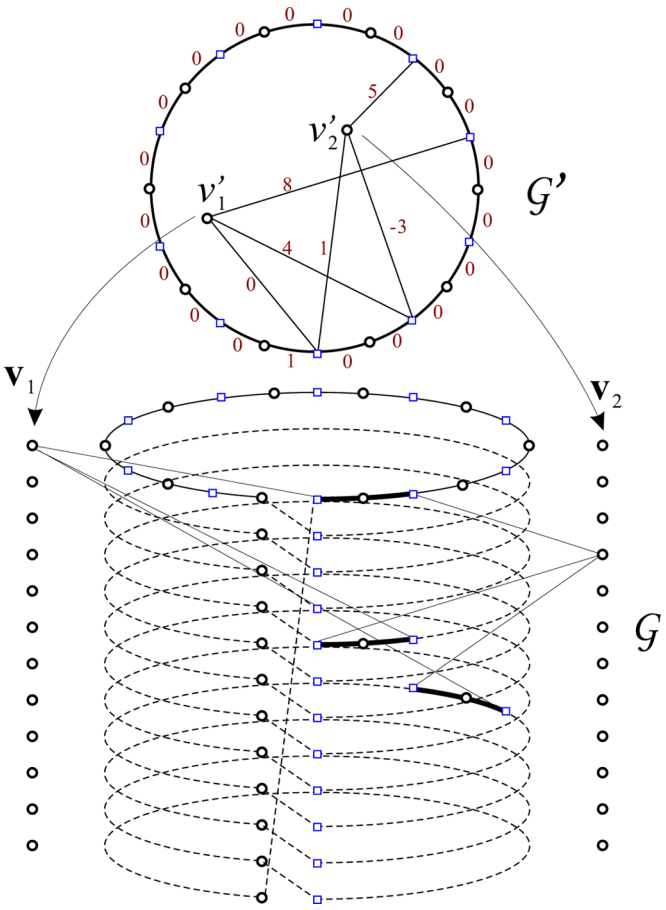

Fig. 3. Code graph based on quasi-cyclic extension and some bad cases of offset values.

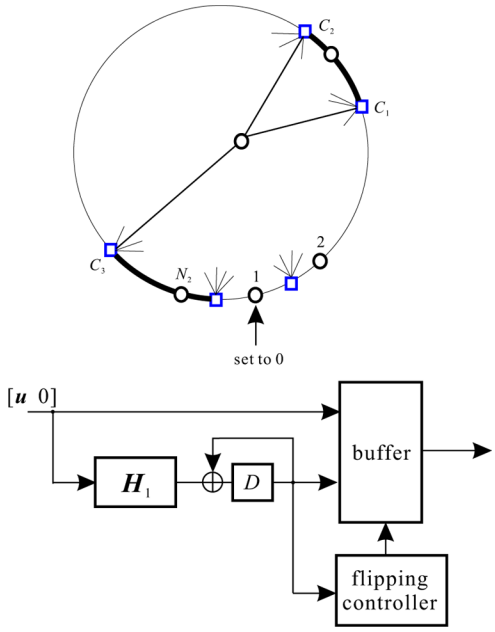

Fig. 4. Encoding of the proposed LDPC code.

in a twisting manner according to the value of $f(e')$. For example, if $f(e') = 2$, then $e_1 = (v_1, u_3), e_2 = (v_2, u_4), \ldots, e_n = (v_n, u_2)$.

All edges on the tail-bited loop have an offset value of 0, except one that has a unit offset. In this way, an n-fold longer loop is formed. In terms of the parity-check matrix, it is similar to generating a large bi-diagonal submatrix. The choice of offset values for the edges in $\mathcal{G}'$ is crucial to the property of the code. Some offset values may lead to low-weight codewords or short cycles in $\mathcal{G}$. In the following, we describe how to choose these values subject to the constraints introduced in Section III-A.

The ith group of degree-3 bit nodes, denoted as $\boldsymbol{v}_i$ for $1 \le i \le N'_3$, is characterized by the three check nodes connected to v'_i in $\mathcal{G}'$, denoted as $\boldsymbol{c}'_i$, and the three corresponding offset values, denoted as $\boldsymbol{f}_i$. Given $\{\boldsymbol{c}'_i\}$, our goal is to select N'_3 3-tuples $\boldsymbol{f}_i, 1 \le i \le N'_3$, such that the resultant graph, $\mathcal{G}$, complies with the three constraints of Section III-A. Due to the quasi-cyclic property, the complexity of this search is of the order $O({N'_3}^2)$, compared to $O({N_3}^2)$, for randomly constructed graphs, with $N_3 = N'_3 \cdot n$.

For example, for a degree-3 node v' in $\mathcal{G}'$, with $\boldsymbol{f} = (f_1, f_2, f_3)$ for its three edges, if $f_1 = f_2$, then Constraint 1 of Section III-A may not be valid for each bit node in the group $\boldsymbol{v}$. The reason is that any bit node in $\boldsymbol{v}$ has two edges corresponding to f_1 and f_2 connected to two check nodes of the same level (see Fig. 3), and therefore the two check nodes may be close to each other. Even if $f_i - f_j$ is equal to 1 or $-1 \bmod n$, the two check nodes, now on two adjacent levels, can still be close to each other, since two adjacent levels are connected too, and it is possible they have check nodes close to each other. If $f_i - f_j$ is not equal to $0, 1$, or $-1 \bmod n$ for any $1 \le i, j \le 3, i \ne j$, then any two check nodes connected to a bit node in $\boldsymbol{v}$ are separated by at least N'_2 degree-2 bit nodes, and Constraint 1 in Section III-A holds with $d_1 > N'_2$.

To illustrate a choice of offset values leading to cases violating Constraint 2 of Section III-A, we consider two degree-3 bit nodes in $\mathcal{G}'$, denoted as v'_1 and v'_2 in Fig. 3. With the chosen offset values labeled on the edges, the code has a codeword of weight 5, because the ith bit in $\boldsymbol{v}_1$ and the $(i+3)$th $\bmod n$ bit in $\boldsymbol{v}_2$ are not widely separated. For brevity, we only plot the case for $i = 1$ in Fig. 3. In addition, it is also possible that two bits within the same group, say $\boldsymbol{v}_1$, are not widely separated due to an inappropriate choice of the three offset values for v'_1.

We also need to pay attention to short cycles as discussed in Constraint 3 of Section III-A. We only consider short cycles consisting of degree-2 and degree-3 bit nodes. Hence, there may exist short cycles with two degree-3 bit nodes involved, and the two bit nodes may be from one group of bit nodes, or from two different groups of bit nodes.

C. Encoder Design

When the number of degree-2 bit nodes is equal to the number of check nodes on the graph, i.e., $N_2 = N - K$, the restricted form of the code admits a simple encoder computation that is linear in the code length. For encoding purpose, all the degree-2 bits are chosen to be the parity bits and all other bits of higher degrees are information bits. The introduction of "tailbiting" may seem a nuisance for the sake of encoding, but it is of little consequence. However, we do need to free a degree-2 bit and let a degree-3 bit be a parity bit. For convenience, the first bit on the closed loop is chosen to be the freed degree-2 bit, and is set to be 0 initially. That is equivalent to deleting the first bit from the graph. We suppose that the degree-3 parity bit is connected to parity check c_1, c_2, and c_3, with $c_1 < c_2 < c_3$, as shown in Fig. 4. Initially, we also set the degree-3 bit to zero. Then given the sum of incoming information bits for each check node, the parity bits on the loop are determined in a sequence from 2 to N_2, similarly to the RA code with a rate-1 two-state convolutional encoder, called accumulator. If we find that the first bit and the last bit on the loop cannot satisfy the check between them, i.e., the final state of the convolutional code is not equal to 0, then the degree-3 bit is flipped, as well as all the parity bits between checks c_1 and c_2, and between checks c_3 and N_2. To reduce

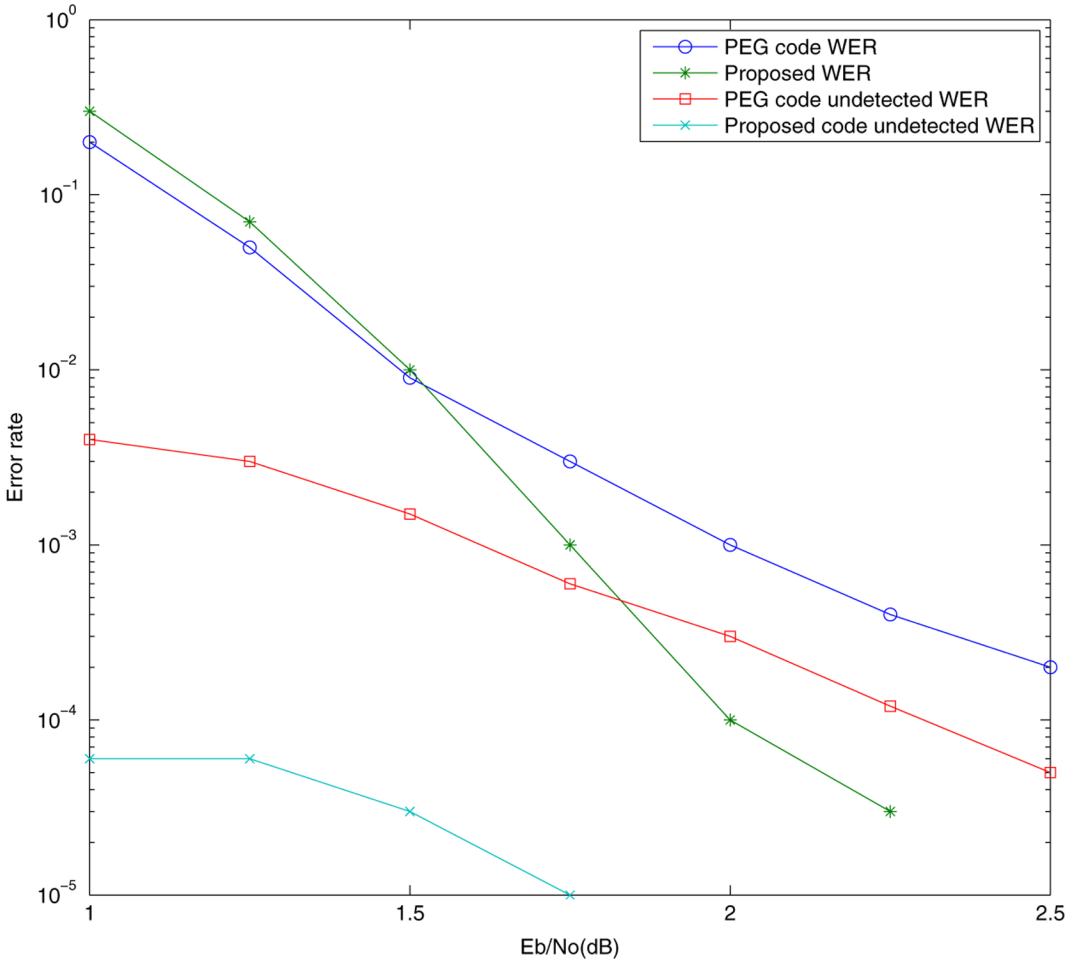

Fig. 5. WER of the $(1728, 864)$ irregular LDPC codes constructed based on PEG and the proposed method.

the number of flipping operations, we can choose a degree-3 parity bit node with a small value $c_2 - c_1 + N_2 - c_3$. The LDPC code is actually an $(N-1, K-1)$ code.

IV. Simulation Results

Fig. 5 depicts the word error rate (WER) of $(1728, 864)$ irregular LDPC codes constructed based on PEG and the proposed method, with the standard BP decoding and a maximum number $I_{\max} = 200$ iterations. The bit and check degree distributions of the PEG-based code are $\lambda(x) = 0.2716x + 0.3333x^2 + 0.3951x^7$ and $\rho(x) = 0.2222x^5 + 0.7778x^6$, respectively. The proposed code is constructed from a small graph with $(N_2', N_3', N_8') = (24, 17, 7)$ and an extension factor $n = 36$, which translates into bit and check degree distributions of $\lambda(x) = 0.3097x + 0.3290x^2 + 0.3613x^7$ and $\rho(x) = 0.5032x^5 + 0.4968x^6$, respectively. Note that (N_2', N_3', N_8') are selected to make the bit degree distribution of the proposed code as close to that of the PEG-based code as possible. The number of degree-2 bit nodes is made to be the same as the number of check nodes in this construction. The construction parameters d_1, d_2, and d_3 are $24, 12$, and 26, respectively. We observe that in the high signal-to-noise ratio (SNR) region, the proposed scheme reduces the error floor by more than one order of magnitude, compared to the PEG-based LDPC code. We also observe that the LDPC code constructed based on the proposed approach has much less undetected errors than the PEG-based LDPC code. The undetected WER of the proposed code is about two orders of magnitude below that of the PEG one.

Fig. 6 depicts the WER of $(1728, 864)$ irregular LDPC codes constructed based on ACE and the proposed method. The simulation is run on an field-programmable gate array (FPGA)-based LDPC simulator with $I_{\max} = 100$. It can be seen that the proposed code outperforms the ACE-based code by a noticeable margin, especially in low-WER regions. At WER $= 10^{-6}$, the gap can be as large as 0.4 dB. Note that the ACE constructed version of the code did not ensure a maximum length tail-biting chain for degree-2 nodes. We also constructed a $(4608, 2304)$ LDPC code and compared the performance with an ACE-based code of the same parameters. The simulation shows that the ACE-based code has a slightly better performance in the WER ranges down to 10^{-4}. However, for the ACE-based code, many undetected error codewords of low weights are collected in the simulation, while for the proposed code, no undetected errors are observed.

TABLE I
Weight Profiles for Three $(1728, 864)$ LDPC Codes

weight	PEG	ACE	proposed
14	72	-	-
15	-	26	-
16	191	29	-
17	323	-	-
18	268	-	9
19	944	36	-

Applying the impulse method of [10], we collected low-weight codewords for the three $(1728, 864)$ codes and record the weight profiles in Table I. The weight profiles for the three codes are obtained under the same simulation conditions. For example, for the ACE-based code, we found 26 codewords of weight 15, 29 codewords of weight 16, and 36 codewords of weight 19. From Table I, we observe that the proposed code exhibits a better minimum distance as well as a thinner profile of

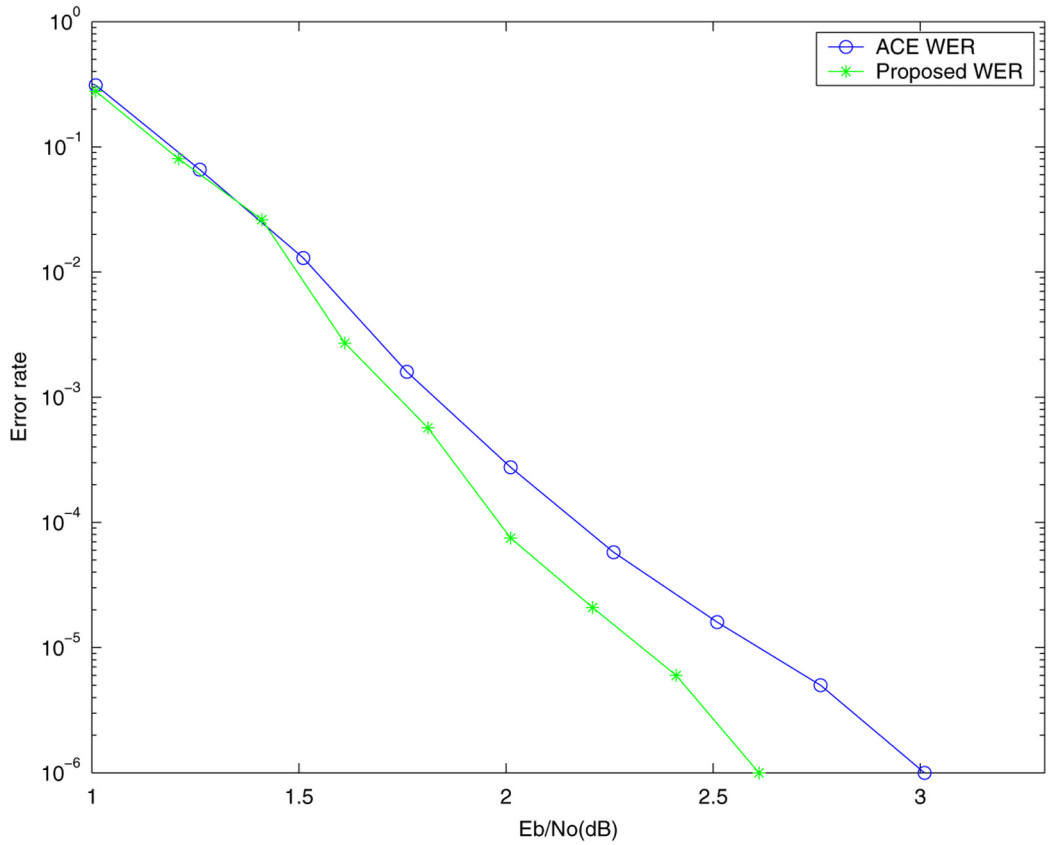

Fig. 6. WER of the $(1728, 864)$ irregular LDPC codes constructed based on ACE and the proposed method.

TABLE II
WEIGHT PROFILE FOR TWO $(4608, 2304)$ LDPC CODES

weight	ACE	proposed
25	95	-
27	-	27
28	156	-
29	253	13
30	59	3
31	235	17
32	108	27
33	319	12
34	792	-

low-weight codewords than the two other codes. This accounts for the better performance of the proposed $(1728, 864)$ code that we observe in simulation. Similarly, Table II shows the weight profiles for the two $(4608, 2304)$ codes. Compared to the ACE-based code, the proposed $(4608, 2304)$ code has a much better weight profile, and conceivably a better performance in low WER regions.

Simulation results show that short LDPC codes constructed with our approach exhibit good performance in the high-SNR region, with low error floors. Furthermore, few undetected errors have been observed for these codes. In this work, constraints related to error words associated with degree-2 and degree-3 nodes have been considered. Additional constraints can be readily introduced if necessary, at the expense of increased complexity in the code design.

ACKNOWLEDGMENT

The authors would like to thank Dr. Christopher R. Jones of the Jet Propulsion Laboratory in Pasadena, CA for valuable discussions, for constructions of comparative ACE codes, and for the simulation results of Fig. 6. The authors are also grateful to Dr. David Declercq for providing all the data in Tables I and II.

REFERENCES

[1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.
[2] T. J. Richardson, A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[3] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. Communication, Control, and Computing*, Monticello, IL, Sep. 2003, pp. 1426–1435.
[4] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.
[5] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.
[6] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth tanner graphs," in *Proc. Globecom 2001*, San Antonio, TX, Nov. 2001, pp. 995–1001.
[7] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.
[8] R. M. Tanner, "On quasi-cyclic repeat-accumulate codes," in *Proc. 37th Allerton Conf. Communication, Control and Computing*, Monticello, IL, Sep. 1999, pp. 249–259.
[9] M. Yang and W. E. Ryan, "Lowering the error-rate floors of moderate-length high-rate irregular LDPC codes," in *Proc. IEEE Int Symp. Information Theory*, Yokohama, Japan, Jun./Ju. 2003, p. 237.
[10] X. Hu, M. Fossorier, and E. Eleftheriou, "On the computation of the minimum distance of low-density parity-check codes," in *Proc. IEEE Int. Conf. Communications*, Paris, France, Jun. 2004, pp. 767–771.