Extrinsic Tree Decoding

Eric Psota
Department of Electrical Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588–0511
Email: epsota24@huskers.unl.edu

Lance C. Pérez, Ph.D.
Department of Electrical Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588–0511
Email: lperez@unl.edu

***Abstract*—A new decoding method, called extrinsic tree decoding, is presented for decoding low-density parity-check codes on modified finite computation trees. The proposed method maintains similar performance to that of existing iterative decoders, while providing a decoding method for which realistic upper bounds can be computed for practical codes.**

I. Introduction

Low-density parity-check (LDPC) codes are capable of near capacity performance when decoded with iterative message-passing decoders such as the sum-product (SP) and min-sum (MS) decoders. These decoders are optimal for codes whose graphical representation is a given by a tree. Unfortunately, practical codes are seldom represented by a tree, and the performance analysis is difficult when the Tanner graph of an LDPC code contains cycles. In his thesis, Wiberg [1] introduced the *computation tree* as an exact model for the SP and MS algorithms for finite iterations. In theory, the computation tree enables one to analyze the performance of the SP and MS decoders at any given iteration. However, the size of the computation tree grows exponentially with the check and variable node degree of the code. For codes with practical variable and check node degrees the growth rate makes analysis intractable even after just a small number of iterations. Thus, it is of practical interest to formulate a decoder whose performance is comparable to that of SP and MS, but which allows for tractable performance analysis on codes with practical variable and check node degrees.

Recent work by Lu, Méasson and Montanari [2] has demonstrated some of the potential advantages of decoding on modified computation trees. In this paper a new decoder called extrinsic tree (ET) decoding is introduced. This new decoder operates by iteratively constructing finite, modified computation trees designed toward minimizing the probability of bit error at the root node. By decoding on these trees, it is possible to reduce the negative effects that cycles in the Tanner graph typically have on decoders. In addition, the performance of ET decoding can be bounded using the weight and multiplicity of the deviation set.

II. Background

The min-sum and sum product algorithms are efficient decoders for LDPC codes with excellent performance. However, the behavior and performance of MS and SP is difficult to characterize analytically for specific codes with cycles. Currently, there are several models for estimating the performance of MS and SP including stopping sets [3], trapping sets [4], pseudocodewords [5] and density evolution [6], [7]. Unfortunately, none of these models is currently capable of bounding the performance of iterative decoding on the binary input additive white Gaussian noise (BIAWGN) channel for a specific code.

Through the use of computation trees and the notion of *valid configurations* on these trees, Wiberg [1] created a perfect model for describing the behavior of MS and SP at finite iterations. Figure 1 shows a Tanner graph representation of a length 3, dimension 2, and minimum distance 2 code, denoted as a $(3, 2, 2)$ code. In this figure, the square nodes represent check nodes and the round nodes represent variable nodes. Figure 2 shows a computation tree rooted at variable node v_1 after two iterations for this code. Wiberg was able to show that the MS and SP decoders are optimal when operating on the computation tree. The MS decoder chooses the minimal cost valid configuration on the tree and the SP decoder choose the highest probability binary value of the root node after considering each valid configuration. A *valid configuration* is any binary assignment to the set of variable nodes such that each check node is connected to an even number of variable nodes assigned a binary 1.

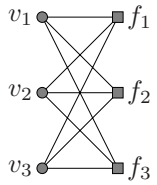


Fig. 1. Tanner graph of a $(3, 2, 2)$ code.

Although the computation tree model is precise, after a small number of iterations it becomes impractical to analyze the performance of specific codes by considering all valid configurations on the computation tree. The number of valid configurations on the computation tree can be computed by treating the computation tree as a Tanner graph. In order to define a Tanner graph given the computation tree, treat all check nodes and variable nodes in the computation tree separately. For example, if multiple copies of variable node v_1 are distributed throughout the computation tree, each copy

978-1-4244-2734-5/09/$25.00 ©2009 IEEE

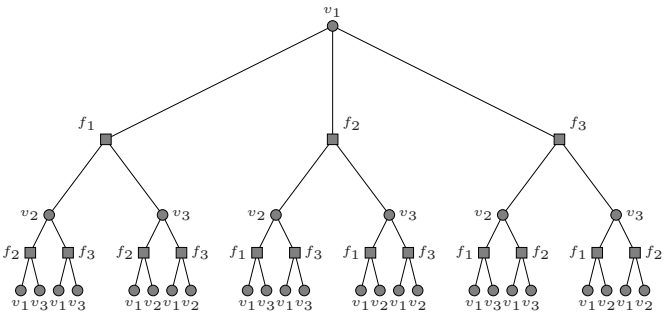


Fig. 2. Computation tree for the $(3, 2, 2)$ code of Figure 1 after 2 iterations.

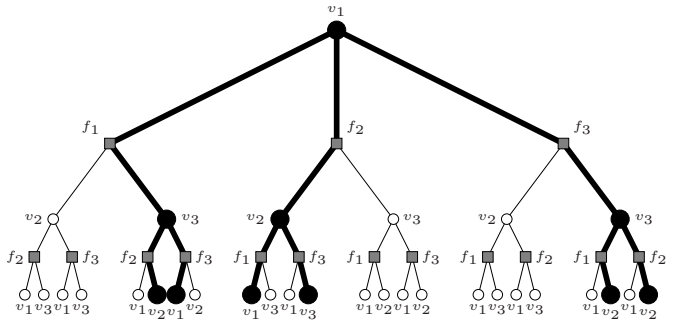


Fig. 3. Example of a deviation on the computation tree.

Iterations	Variable Nodes	Check Nodes	Configurations
1	16	3	8192
2	166	33	$\approx 10^{40}$
3	1666	333	$\approx 10^{401}$

TABLE I
NUMBERS OF NODES AND VALID CONFIGURATIONS ON THE COMPUTATION TREE OF A $(6, 3)$-REGULAR LDPC CODE.

is treated as a unique variable node. After regarding each variable node in the computation tree as unique, it is possible to show that each check node is linearly independent. If the computation tree has N variable nodes and M check nodes, then there are 2^{N-M} valid configurations on the tree. To illustrate the growth rate in the number of valid configurations on the computation tree, first consider an LDPC code where each check node has degree $d_F = 6$ and each variable node has degree $d_V = 3$. These commonly used code parameters define what is known as a $(6, 3)$-regular LDPC code. For this code, Table I shows the number of variable nodes, the number of checks nodes, and number of valid configurations on the computation tree after 1, 2, and 3 iterations. Note that the growth rate is not affected by the length of the code.

In order to simplify the computation tree analysis of MS, Wiberg introduced the idea of *deviations*. A deviation is a set of variable nodes assigned to a binary 1 on the computation tree such that the following two conditions are satisfied:

1) The root node of the tree must be assigned a binary 1.
2) No subset of variable nodes in the deviation form a valid configuration on the tree.

Figure 3 gives an example of a deviation on the computation tree of Figure 2. The larger black variable nodes have been assigned a binary 1, and the white nodes have been assigned a binary 0. Notice that each check node is connected to either two or zero variable nodes assigned a binary 1, and that a change in the binary assignment of any of the variable nodes in the set makes the configuration invalid. Wiberg showed that in order for an error to occur at the root node of a computation tree during min-sum decoding it is necessary, but not sufficient, for at least one deviation on the tree to have negative cost. The probability of error at the root node P_e can thus be bounded by

$$P_e \leq \bigcup_{i \in D} P(B_i),$$

where D is the set of all deviations and B_i is the event that deviation i has negative cost. Since it is generally difficult to compute the union, this bound can be loosened to

$$P_e \leq \sum_{i \in D} P(B_i). \tag{II.1}$$

It can be shown that the number of deviations on $R_{v_i}^{(\ell)}$ is

$$(d_F - 1)^{\sum_{i=1}^{\ell} d_V (d_V - 1)^{i-1}},$$

where $R_{v_i}^{(\ell)}$ is the computation tree of a (d_F, d_V) regular LDPC code, rooted at variable node v_i after ℓ iterations. Table II shows the number of deviations for a $(6, 3)$-regular LDPC code for iterations 1 through 5. Note that, just like the number of valid configurations, the growth rate is not affected by the length of the code. Comparing the total number of valid configurations to the total number of deviations, it is easy to see that deviations are much more practical to enumerate. However, because iterative decoding is typically performed for 100 or more iterations, it is still impractical to use this approach to analyze the performance of MS decoding.

Iterations	# of Deviations
1	125
2	1,953,125
3	4.7684×10^{14}
4	2.8422×10^{31}
5	1.0097×10^{65}

TABLE II
NUMBER OF DEVIATIONS FOR ITERATIONS 1 THROUGH 5 FOR A $(6, 3)$-REGULAR LDPC CODE.

III. EXTRINSIC TREE DECODING

Motivated by both the analytical difficulty and the sub-optimality of MS, a new decoding method known as extrinsic tree (ET) decoding is presented here as an alternative to MS. The primary goals of ET decoding are to:

1) Decode on smaller trees so performance can still be analyzed.

2) Design trees with low probability of error at the root node.

The first step for ET decoding is to build an extrinsic tree for each variable node. After the extrinsic trees are built, decoding operations are performed from the leaf nodes up to the root node. The ET decoder operates on N different trees, where N is the length of the code.

The process for building an extrinsic tree rooted at a particular variable node v_i is as follows. First, let $N(v)$ be the neighborhood of an abitrary variable node v in the Tanner graph, and let $N(f)\backslash v$ be the neighborhood of an arbitrary check node f excluding variable node v.

1) Root the extrinsic tree at variable node v_i. Set the level of the tree to $\ell = 0$.
2) For each v_k in the tree at level ℓ
   a) If check node $f_j \in N(v_k)$ is not the parent node of v_k, connect f_j to v_k, and connect all other variable nodes $v_l \in N(f_j)\backslash v_k$ to f_j at level $\ell + 1$.
   b) Find the minimum weight deviation on the extrinsic tree. If the minimum weight is increased or kept the same, keep f_j and its corresponding variable node. Otherwise, eliminate f_j and its corresponding variable nodes from the extrinsic tree.
3) $\ell = \ell + 1$. Return to Step 2.

It is important to note that there might be multiple copies of a variable node v_i in the extrinsic tree. If this is the case, the log-likelihood ratio (LLR) cost λ_i is split evenly among the copies of node v_i. Figure 4(a) illustrates an example where there are three copies of variable node v_2 in the tree, and Figure 4(b) shows the distribution of cost λ_2 among the nodes. In this example, all other nodes v_i with only a single copy in the tree would be assigned an unscaled cost λ_i.

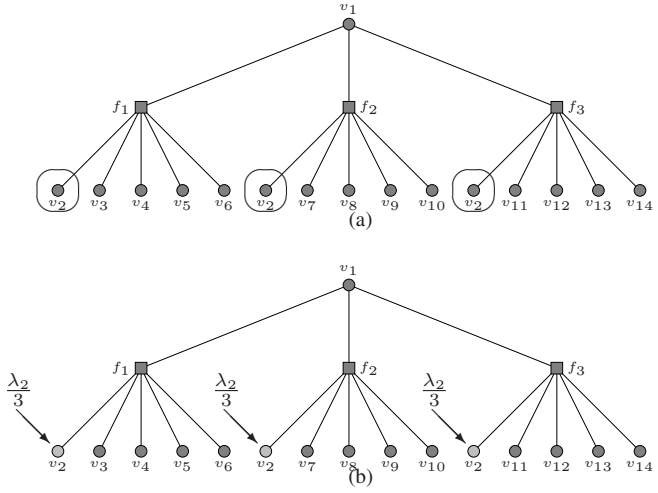


Fig. 4. Cost scaling on the extrinsic tree.

With this new scaling in mind, it is now necessary to explain what is meant by the weight of a deviation. Consider the binary-input, additive, white, Gaussian noise (BIAWGN) channel given by the following:

- Binary codeword: $\mathbf{c}$
- Modulated codeword ($0 \rightarrow -1$, $1 \rightarrow +1$): $\mathbf{x}$
- AWGN noise (variance σ^2): $\mathbf{n}$
- Received vector: $\mathbf{y} = \mathbf{x} + \mathbf{n}$
- Log-likelihood ratio cost: $\boldsymbol{\lambda} = -\mathbf{y}$
- Probability of a sending $x_i = -1$, and receiving $-y_i < 0$:

$$\int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y_i+1)^2}{2\sigma^2}} dy$$

Now, consider a deviation containing the set of variable nodes $\{v_1, \ldots, v_w\}$, where there are a_i copies of each variable node v_i in the deviation. There are b_i copies of each variable node v_i in the entire extrinsic tree. The probability that each node in the deviation was transmitted as a -1 and that the deviation has cost

$$\sum_{i=1}^{w} \frac{a_i}{b_i}(-y_i) < 0$$

can be found by looking at the probability that a single Gaussian random variable with distribution

$$\mathcal{N}\left(\sum_{i=1}^{w} \frac{a_i}{b_i} \ , \ \sum_{i=1}^{w} (\frac{a_i}{b_i}\sigma)^2\right)$$

is less than zero. This random variable can be rescaled resulting in the distribution

$$\mathcal{N}\left(\left(\frac{\sum_{i=1}^{w} \frac{a_i}{b_i}}{\sqrt{\sum_{i=1}^{w} (\frac{a_i}{b_i})^2}}\right)^2 , \left(\frac{\sum_{i=1}^{w} \frac{a_i}{b_i}}{\sqrt{\sum_{i=1}^{w} (\frac{a_i}{b_i})^2}}\right)^2 \sigma^2\right).$$

Note that the mean

$$\left(\frac{\sum_{i=1}^{w} \frac{a_i}{b_i}}{\sqrt{\sum_{i=1}^{w} (\frac{a_i}{b_i})^2}}\right)^2 \tag{III.1}$$

of the distribution is precisely the deviation weight used when constructing the extrinsic tree.

Consider a deviation where where $(v_1, \ldots, v_w)$ corresponds to a codeword in the original Tanner graph, and each of the $(b_1, \ldots, b_w)$ copies of the variable nodes in the extrinsic tree is contained in the deviation. Then each $\frac{a_i}{b_i} = 1$, and Equation III.1 reduces to

$$\left(\frac{\sum_{i=1}^{w} 1}{\sqrt{\sum_{i=1}^{w} 1^2}}\right)^2 = \frac{w^2}{w} = w$$

which is just the Hamming weight of the codeword. Thus, the probability of a codeword deviation error is determined by the Hamming weight of the codeword, which is a desirable property made possible by the choice of $(b_1, \ldots, b_w)$ as cost scalars.

Unfortunately, with larger codes it becomes computationally impractical to consider each deviation in each of the extrinsic trees. So, it is worth considering ways of maximizing the minimum deviation weight without using an exhaustive search. For example, adding a check with all-new variable nodes to the tree cannot decrease the weight of an existing deviation. A further conjecture is that adding multiple checks at the lowest level in the tree with all-new variable nodes (allowing multiple copies of the same variable node among the new checks) cannot decrease the weight of an existing deviation. With these observations, the extrinsic tree building process can be changed so that a check and its corresponding variable nodes are added to level ℓ if and only if none of the variable nodes have appeared on any previous level $< \ell$ in the tree. For the remainder of the paper, only extrinsic trees constructed with these new rules are considered.

Figure 5 shows the Tanner graph of a $(7, 2, 3)$ LDPC code, and Figure 6 demonstrates the extrinsic tree building process for the tree rooted at variable node v_2. The extrinsic tree in Figure 6(a) shows the tree after only the first layer of checks and variable nodes was added. After the first layer, the variable nodes v_1, v_2, and v_3 exist in the tree. The second layer of nodes is added in Figure 6(b). Check node f_2 and variable node v_3 were not added to the tree below v_1 because v_3 already exists in a previous layer. A similar situation occurs for the nodes below v_3. However, the check node f_4 and variable nodes v_5 and v_6 are added because these variable nodes do not appear anywhere in previous layers of the tree. Finally, the last layer of nodes is added to the tree in Figure 6(c) because the variable node v_7 does not appear in a previous layer.

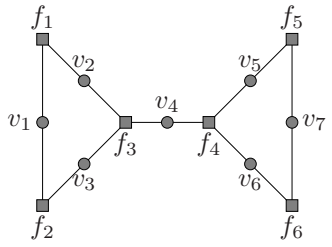
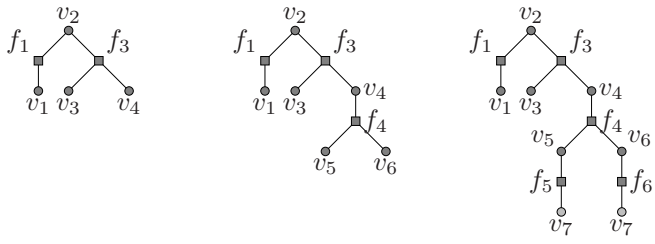


Fig. 5. Tanner graph of a (7,2,3) LDPC code.



(a) Level 1 (b) Level 2 (c) Level 3

Fig. 6. Extrinsic tree rooted at variable node v_2.

The process of building the extrinsic trees only has to be performed once before decoding takes place. After the extrinsic trees are built for each variable node, their size and structure is set and does not change during decoding. The first step in the decoding process is to assign scaled LLR costs to each of the variable nodes in the extrinsic tree. For example, on the extrinsic tree shown in Figure 6(c) the set of LLR costs $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \frac{\lambda_7}{2}\}$ should be assigned to the variable nodes $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. Variable node v_7 receives a scaled version of the LLR because there are multiple copies of v_7 in the tree.

Once LLR costs have been assigned to each of the variable nodes, MS decoding [1] operations are performed from the bottom level of the extrinsic tree up to the the root node. First, let the message m_v at each leaf node in the extrinsic tree be initialized to the LLR cost assigned to that node. Also, let $C(f)$ denote the children nodes of check node f, and let $C(v)$ be the children nodes of variable node v. At each check node, the message

$$m_{f_i} = \left(\prod_{v_k \in C(f_i)} \operatorname{sgn}(m_{v_k}) \right) \left(\min_{v_k \in C(f_i)} |m_{v_k}| \right)$$

is computed from the costs of its children node(s). Next, the variable node message

$$m_{v_i} = \lambda_i + \sum_{f_k \in C(v_i)} m_{f_k}$$

is computed from the check node message(s) of its children. This process is continued up the tree until the message $m_{v_{root}}$ has been computed at the root node of the extrinsic tree. This message represents the difference in cost between the minimum-cost configuration assigning a binary 0 at the root node and the minimum-cost configuration assigning a binary 1 at the root node. This final cost is then quantized using

$$\hat{c}_{root} = \begin{cases} 0 & \text{if } m_{v_{root}} > 0 \\ 1 & \text{if } m_{v_{root}} < 0 \end{cases}$$

to determine the decoded binary value of the root node. The set of binary root node values assigned to each of the extrinsic trees is the final codeword estimate given by the decoder.

Figure 7 shows the decoding performance of the ET decoding process compared to that of regular MS decoding at iterations 2 and 400 on a $(6, 3)$-regular LDPC with length $N = 40$. It is interesting to note that the performance of ET decoding is better than MS decoding after 2 iterations at high SNR, since the extrinsic trees each contain fewer nodes than the computation tree after 2 iterations. The MS decoder with 400 iterations performs better than ET decoding until the SNR reaches 10.5 dB.

In order to demonstrate that the deviation bounds given in (II.1) can be applied to ET decoding, Figure 8 shows the simulated probability of error on the tree rooted at v_1 and the corresponding upper bound computed from enumerating the deviations. The bound comes very close to the simulated performance after about 8.0 dB. At this point, the performance of the decoder is largely dominated by the low-weight deviations on the extrinsic tree.

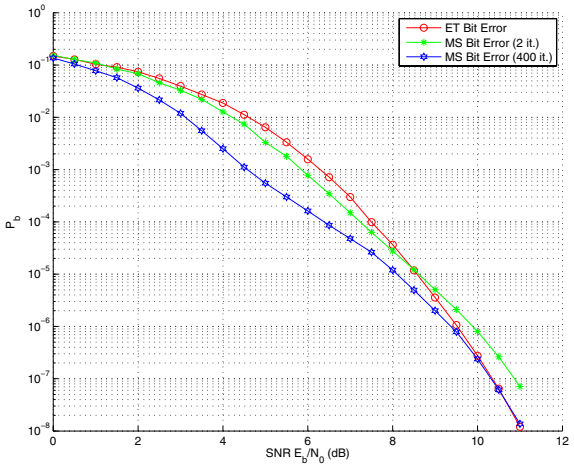


Fig. 7. Probability of bit error for MS and ET decoding for a (6,3)-regular (40,20) LDPC code (MS iterations 2 and 400).

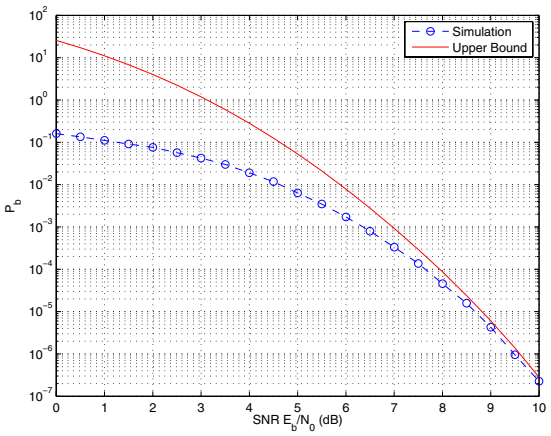


Fig. 8. Probability of bit error for ET decoding (without normalization) of v_1 of the $(40, 20)$ LDPC and the corresponding upper bound.

IV. Conclusion

In this paper, a decoder has been developed for LDPC codes that is both non-iterative and still possible to upper-bound. It has also been demonstrated that ET decoding is capable of exceeding the performance of MS at high SNR.

While the process of building extrinsic trees can be computationally complex, the trees only need to be constructed once for each code. One advantage of ET decoding is that the decoding operations from leaf nodes to the root node can be done in parallel with the required number of clock cycles determined by the depth of the tree, which is considerably smaller than that of regular MS decoding even after a very small number of iterations.

Future work includes devising additional techniques for reducing the computational complexity of deviation checking for large trees. Also, by exploiting the structure of the new decoder, it might be possible to design LDPC codes to perform well with this decoder. Another area of interest is applying normalization to the messages passed from leaf nodes to the root node in such a way that the weight of the minimum weight deviations might be increased.

References

[1] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. thesis, Linköping University, Linköping, Sweden, 1996.

[2] Y. Lu, C. Méasson, and A. Montanari, "TP decoding," in *Proceedings of the 45th Annual Allerton Conference on Communication, Control, and Computing*, September 2007.

[3] G. D. Forney, Jr., R. Koetter, F. R. Kschischang, and A. Reznik, "On the effective weights of pseudocodewords for codes defined on graphs with cycles," in *Codes, systems, and graphical models (Minneapolis, MN, 1999)*, vol. 123 of *IMA Vol. Math. Appl.*, pp. 101–112. Springer, New York, 2001.

[4] D. J. C. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, 2003.

[5] N. Axvig, D. Dreher, K. Morrison, E. Psota, L. C. Pérez, and J. L. Walker, "Analysis of connections between pseudocodewords," Submitted to *IEEE Transactions on Information Theory*, March 2008.

[6] T. Richardson and R. Urbanke, "The capacity of low-density parity check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, February 2001.

[7] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.