Design of Flexible LDPC codes for Scalable Energy requirements

Surajkumar Harikumar

***Abstract*—In this paper, we introduce propose a method of designing flexible LDPC codes, which can give scalable performance by check node additions. As shown in [1], the energy consumed by the decoder circuit increases as we approach lower probabilities of error. Designing codes which can scale in performance enables us to use lower energy at short distances and achieve high performance for long distance communication. We introduce a framework for computing the average change in the degree distribution resulting from adding check nodes. We then use a variable number of check node additions to give an exponential improvement in probability of error (under density evolution for the Gallager B algorithm). We propose a framework for connecting check nodes between instances of local decoders, and investigate its performance and energy requirement.**

***Index Terms*—LDPC codes, Flexible codes, Bit meters, Energy efficient coding**

I. Introduction

II. Background

ROBERT Gallager introduced a new class of codes, named *Low Density Parity Check Codes* (LDPC Codes) in his thesis [2] in 1961. This class of codes relies on sparse-parity check matrices and iterative decoding by message-passing, to achieve high performance. In his original work, Gallager introduced what we now call *regular*-LDPC codes. A regular (d_v, d_c) LDPC code has exactly d_v ones in each column, and d_c ones in each row. Considering the Tanner Graph [3] representation of LDPC codes, every variable-node is connected to d_v check nodes, and every check-node is connected to d_c variable nodes. Gallager gave methods for constructing these codes, and proposed iterative decoding algorithms (which we now call the Gallager A and Gallager B algorithms).

Irregular LDPC codes have been the subject of much study after they were shown to be capacity achieving on the BEC [4]. Previous works [5],[6],[7] expand the code definitions and algorithms of regular LDPC codes to irregular graphs. We shall summarize some of the relevant definitions, notations, and density-evolution formulae here.

An irregular-LDPC code is one where the variable and check nodes do not necessarily have the same number of edges leaving the nodes. Irregular-LDPC codes are commonly characterized by their *Degree-Distribution*, which is polynomial describing the fraction of nodes / edges of a certain degree. LDPC codes can be expressed using either the *Node Degree-Distributions* and *Edge Degree-Distributions*, which are easily related to each other. The Edge Degree-Distribution (λ, ρ) of an LDPC code tells us the fraction of edges connected to a node of a given degree. More specifically,

$$\lambda(x) = \sum_i \lambda_i x^{i-1}, \qquad \rho(x) = \sum_i \rho_i x^{i-1} \tag{1}$$

where $\lambda_i(\rho_i)$ denoted the fraction of edges incident to a variable (check) node of degree i. Alternatively, $\lambda_i(\rho_i)$ can be viewed as the probability that any random edge in the graph is connected to a variable (check) node of degree i. So if the code has M edges and N variable nodes, the number of variable nodes of degree i is $M\lambda_i/i$. So the number of variable nodes, is given by $N = M\sum_i \lambda_i/i$. So the number of edges M is given by

$$M = \frac{N}{\sum_i \lambda_i/i} = \frac{N}{\int_0^1 \lambda(x)} \tag{2}$$

The number of check nodes of degree i is given by $M\rho_i/i$. So the total number of check nodes is given by $M\sum_i \rho_i/i = N\frac{\int_0^1 \rho(x)}{\int_0^1 \lambda(x)}$. It follows that the design rate of the code is

$$r = 1 - \frac{N_{check}}{N_{variable}} = 1 - \frac{\int_0^1 \rho(x)}{\int_0^1 \lambda(x)} \tag{3}$$

For a regular-(d_v, d_c) LDPC code, $\lambda(x) = x^{d_v-1}$, and $\rho(x) = x^{d_c-1}$.

Luby *et al.*, investigated the performance of irregular LDPC codes for the BSC in [8]. Of particular interest to us is the extension of the Gallager B to irregular graphs, which is known to perform very well for regular graphs and single-bit messages. In this algorithm, a threshold $b_{i,j}$ is used to make bit-flip decisions in iteration i for a variable node of degree j. Having this extra parameter lets us design codes which perform much better. Accounting for the irregularity of the code in the Gallager B algorithm, the density evolution formula giving us the fraction of incorrect messages p_i in the ith iteration is given by

$$p_{i+1} = p_0 - p_0\left[\sum_{j=1}^{d_v}\lambda_j \sum_{t=b_{i+1,j}}^{} \binom{j-1}{t}\alpha^t\beta^{j-1-t}\right] + (1-p_0)\left[\sum_{j=1}^{d_v}\lambda_j \sum_{t=b_{i+1,j}}^{} \binom{j-1}{t}\alpha^{j-1-t}\beta^t\right] \tag{4}$$

where $\alpha = \frac{1+\rho(1-2p_i)}{2}$, and $\beta = \frac{1-\rho(1-2p_i)}{2}$. The term $b_{i+1,j}$ can be chosen to minimize p_{i+1}, that is, $b_{i+1,j}$ is the smallest integer such that

$$\frac{1-p_0}{p_0} \leq \left(\frac{1+\rho(1-2p_i)}{1-\rho(1-2p_i)}\right)^{2b_{i+1,j}-j+1}$$

which gives us,

$$b_{i+1,j} = \left\lceil \left(j - 1 + \frac{log(1-p_0)/p_0}{log\frac{1+\rho(1-2p_i)}{1-\rho(1-2p_i)}} \right) /2 \right\rceil \quad (5)$$

This gives us a good analytical description of the Gallager B algorithm, and allows us to run density evolution over any irregular LDPC code (reliable upto its girth). This formula gives us an initial estimate as to how our code constructions perform, and are a preliminary to the actual decoder simulations.

III. Check Node additions and tradeoffs

In this section we investigate the effect of adding a check node to the decoder. We first list out the — likely to change by check node additions. We then give an analytical method for computing the expected change in the edge-degree distribution resulting from adding a single check node for degree d. This considers the change in degree-distribution from adding a single edge to a variable node of degree i, and uses the probability of the edge connecting to a degree i variable node. This is repeated d-times to get the expected degree-distribution after the check node addition.

We compare how this expected (average) degree distribution performs compared to a random edge addition to a regular-LDPC code. We then give an approximation to the above formula, which allows simultaneous addition of all d-edges. We show that if $d << N$, (N being the length of the codeword), the approximation performs close to both the average, and the random scenario.

A. Effects of adding a check node

Intuitively, adding a check node reduces the probability of error by any decoding algorithm. More check nodes creates more redundancy of information, and this is turn yields better performance. From the standpoint of a message-passing decoding algorithm, check-nodes facilitate more flow of information. More information enables a good decoding algorithm to make better decisions, and thus reduces the probability of error.

Adding check nodes reduces the rate of the code. Since most practical applications require the code to be above a certain rate, it is easy for us to design a higher-rate base code, and add check nodes as long as we stay within the desired rate.

The girth (length of the smallest cycle) of the code is likely to decrease if we add check nodes to an existing LDPC code. More connections could cause a lower girth, which reduces the independent decoding depth, and thus the performance of the LDPC code. This is one of the factors we seek to correct using our proposed framework in Section IV-C). Adding connections between independent local decoders is much less likely to change the girth as compared to adding connections to a single decoder.

Finally, adding check nodes entails adding edges. Adding edges will increase the wire-length and the information flow at the decoder. This in turn will lead to more *bit-meters* , and a higher energy consumption at the decoder. In [1], it was shown that the bit-meters consumed by the circuit is lower bounded by $B.M = \Omega(log\frac{1}{P_e})$. So, an exponential decrease in P_e will involve atleast a linear increase in the number of *bit-meters*. We also introduce an approximation to the above formula, and we compare it to the other 2.

B. Expected Degree-Distributions

Adding a check node to an existing graph causes a change in the degree distribution of the LDPC code. The change in the degree distribution, alongside the change in the number of edges and/or nodes in the decoder, will affect the performance (both by density evolution and the actual decoder performance). We consider the addition of a single check node of degree-d to an existing (λ, ρ) irregular-LDPC code, with M-edges and N codeword bits.

Lemma 1. *Let $\hat{\rho}(x)$ be the check-node edge-degree distribution after the addition of a single check node of degree d. Then*

$$\hat{\rho}(x) = \frac{M}{M+d}\rho(x) + \frac{d}{M+d}x^{d-1} \quad (6)$$

Proof: Recall that $\rho_i x^{i-1}$ represents the fraction of edges connected to a check node of degree i. So the actual number of edges (connected to a check node of degree i) is $M\rho_i$. Now, we are add d-edges to the new-check node of degree d. So the number of edges connected to the degree d-nodes is $M\rho_d + d$. The other terms stay the same. There are $M + d$ edges in the new graph. Equating the edge terms gives us, $(M + d)\hat{\rho}(x) = M\rho(x) + dx^{d-1}$, or

$$\hat{\rho}(x) = \frac{M}{M+d}\rho(x) + \frac{d}{M+d}x^{d-1}$$

■

While change in the check node edge-degree distribution is trivial, the variable node equivalent is not as straightforward. Depending on how the edges are added, the resulting degree distributions could be very different. So, we rely on a probabilistic model, where we consider the addition of one edge at a time. We assume each edge is equally likely to be connected to any of the variable nodes, and that the addition of each edge is independent.

Let $\lambda^{(0)}(x) = \lambda(x), \rho^{(0)}(x) = \rho(x), M^{(0)} = M, d_v^{(0)}$ represent the initial parameters of the code, and $\lambda^{(l)}(x), \rho^{(l)}(x), M^{(l)}, d_v^{(l)}$ be the parameters after the addition of l edges. It is very easy to see that $M^{(l)} = M^{(l-1)} + 1 = M^{(0)} + l$. The $\rho^{(l)}$ term is also fairly easy to compute.

d_v represents the highest order term in $\lambda(x)$. The addition of an edge can potentially cause the maximum order to increase by 1 (addition of an edge to the highest order node). So, $d_v^{(l)} = d_v^{(l-1)} + 1 = d_v^{(0)} + l$. Note that this does not rule out the possibility of multiple edges between the same variable and check node. As we will show later, if $d << N$, this event is very unlikely to occur, and so this formula will converge to the exact case. Ruling out multi-edges becomes very hard for a generic edge addition to an irregular graph, and it is easier to add edges independently.

The $\lambda^{(l)}(x)$ term depends on the exact connection of edges, so we compute instead the average value $\hat{\lambda}^{(l)}(x)$ over all possible connections.

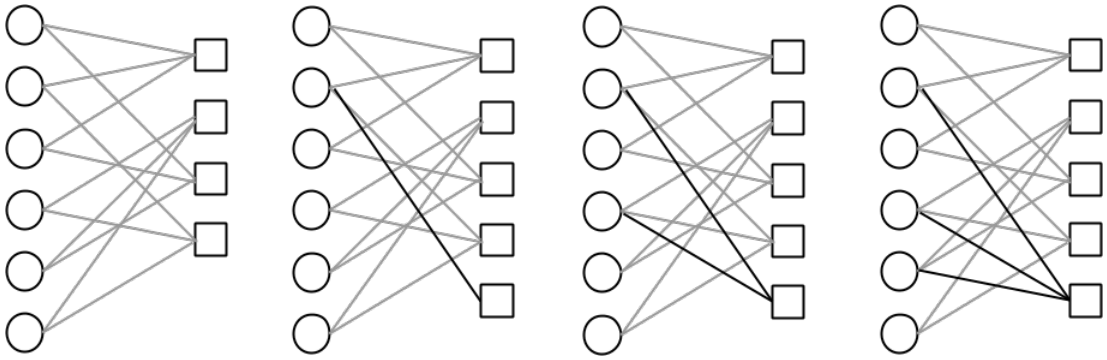


Fig. 1: Independent random addition of edges to a new check node

Lemma 2. *For a given $\lambda^{(l)}(x) = \lambda(x), M^{(l)} = M, N, d_v^{(l)} = d_v$, the coefficients of the terms in $\hat{\lambda}^{(l+1)}(x) = \hat{\lambda}'(x)$ are given as*

$$\hat{\lambda}'_i = \begin{cases} \frac{M}{M+1}\lambda_0\left(1-\frac{1}{N}\right) & , i = 1 \\ \frac{M}{M+1}\left[\lambda_i\left(\frac{N-1}{N}\right) + \frac{1}{N}\lambda_{i-1}\left(\frac{i}{i-1}\right)\right] & , 2 \le i \le d_v \\ \frac{M}{M+1}\frac{1}{N}\lambda_{d_v}\left(\frac{1+d_v}{d_v}\right) & , i = d_v+1 \end{cases} \tag{7}$$

Proof: We know that the number of variable nodes of degree k is $M\lambda_k/k$. From (2) ,we know that the total number of variable nodes is $N = M\sum_k \lambda_k/k$. We have assumed the new edge is equally likely to go to any of the variable nodes. So the probability that a new random edge connects to a variable node of degree k is

$$Pr_k = \frac{M\lambda_k/k}{M\sum_k \lambda_k/k} = \frac{M}{N}\lambda_k/k \tag{8}$$

If the edge connects to a node of degree k, the node now has a degree $k+1$. So, we subtract kx^{k-1} and add $(k+1)x^k$ edges to the (scaled) degree distribution. Also, the new number of edges is $M+1$. So, for the case when the edge connects to a node of degree k, the new edge-degree distribution is

$$\lambda^k(x) = \frac{M}{M+1}\lambda(x) + \frac{1}{M+1}\left[(k+1)x^k - kx^{k-1}\right] \tag{9}$$

Combining (8) and (9), we can find the expected change in degree distribution from the addition of the new edge.

$$\begin{aligned} \hat{\lambda}'(x) = E[\lambda^k(x)] &= \sum_{k=1}^{d_v} \lambda^k(x) Pr_k \\ &= \frac{M}{M+1}\lambda(x)\frac{\sum_k \lambda_k/k}{\sum_k \lambda_k/k} \\ &\quad + \frac{1}{M+1}\left[\frac{\sum_k \lambda_k/k}{N/M}(k+1)x^k - \frac{\sum_k \lambda_k/k}{N/M}kx^{k-1}\right] \\ &= \frac{M}{M+1}\left[\lambda(x) + \frac{1}{N}\sum_k \lambda_k\left[\frac{k+1}{k}x^k - x^{k-1}\right]\right] \end{aligned} \tag{10}$$

Noting that $\lambda(x) = \sum_k \lambda_k x^{k-1}$, and that $\sum_k \frac{\lambda_k}{k}x^k = \int_0^x \lambda(x')dx'$, we can re-write (10) as

$$\begin{aligned} \hat{\lambda}'(x) &= \frac{M}{M+1}\left[\lambda(x) + \frac{1}{N}\left[x\lambda(x) + \int_0^x \lambda(x')dx' - \lambda(x)\right]\right] \\ &= \frac{M}{M+1}\left[\frac{N-1}{N}\lambda(x) + \frac{1}{N}\left[x\lambda(x) + \int \lambda(x)dx\right]\right] \end{aligned} \tag{11}$$

This alternate form for the expected degree distribution will help us in the next section, where we approximate the degree distribution.

Now, we note that $\sum_{k=1}^{d_v} \lambda_k x^k = \sum_{k=2}^{d_v+1} \lambda_{k-1}x^{k-1}$. By definition, $\hat{\lambda}'(x) = \sum_{k=1}^{d_v+1} \hat{\lambda}'_k x^{k-1}$. Putting these in (10), we get

$$\begin{aligned} \hat{\lambda}'(x) &= \sum_{k=1}^{d_v+1} \hat{\lambda}'_k x^{k-1} \\ &= \frac{M}{M+1}\left[\frac{N-1}{N}\lambda(x) + \frac{1}{N}\sum_{k=2}^{d_v+1} \lambda_{k-1}\frac{k}{k-1}x^{k-1}\right] \\ &= \frac{M}{M+1}\left[\frac{N-1}{N}\sum_{k=1}^{d_v} \lambda_k x^{k-1} + \frac{1}{N}\sum_{k=2}^{d_v+1}\frac{k}{k-1}\lambda_{k-1}\right] \end{aligned}$$

Equating each of the k terms separately, we obtain the formula given in (7). ∎

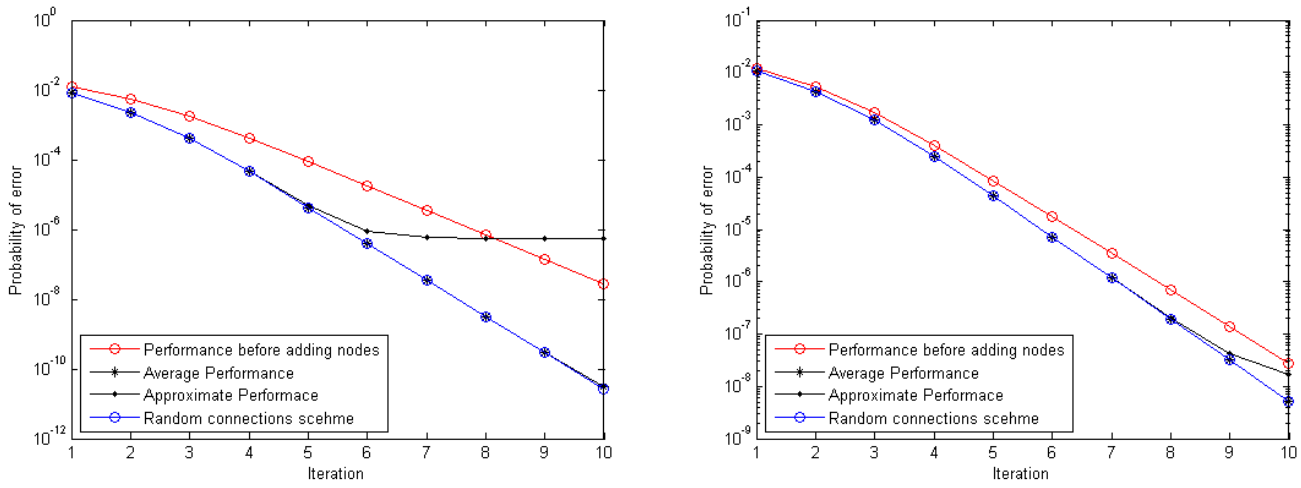
Applying (7) d-times gives us the average resulting variable-node-degree-distribution. We can use this along with (6) to obtain a measure of average performance of the LDPC code after the addition of a single check node. We can apply this multiple times to simulate adding many check nodes.

As mentioned earlier, the proof does not prevent the occurrence of multiple-edges between a single variable and the new check node. As we will show later, if $d << N$, the approximate degree distribution is quite close to the average one given here, and the multi-edge terms are negligible.

C. Approximate Degree Distributions

In the previous section we found the exact check-node degree-distribution from adding a new check node (6), and an average measure of the variable-node degree-distribution (7). The latter was defined recursively, by adding one edge at a time. This is very easy to implement computationally, though for the purpose of analysis, it a more straightforward relationship would give more insight into the problem. This is what we seek to do in this section, developing an approximate measure for the variable node degree distribution after adding d-edges.

Lemma 3. *Consider the addition of d-edges to a given (λ, ρ) irregular-LDPC code of length N, with M edges and*



(a) N=1000

(b) N=4000

Fig. 2: Comparison of the average, approximate and random check node additions performed by adding 100 check nodes to an initially regular-(3,6)-LDPC code with lengths as specified. Density evolution for Gallager B was run to 10 iterations, with $p_0 = 0.02$

a maximum variable node degree d_v. *For* $d << N$ *,the coefficients of the terms in* $\hat{\lambda}^{(d)}(x) = \hat{\lambda}'(x)$ *are given as*

$$\hat{\lambda}'_i = \begin{cases} \lambda_0 \left[1 - \frac{d(N+M)}{N(M+1)}\right] & , i = 1 \\ \lambda_i \left[1 - \frac{d(N+M)}{N(M+1)}\right] + \frac{dM}{N(M+1)} \lambda_{i-1} \left[\frac{i}{i-1}\right] & , 2 \leq i \leq d_v \\ \frac{dM}{N(M+1)} \lambda_{d_v} \left[\frac{d_v+1}{d_v}\right] & , i = d_v + 1 \end{cases} \tag{12}$$

Proof: See Appendix A ■

D. Comparison of Degree Distributions

Intuitively, the average performance obtained by adding check nodes to an initially regular-LDPC code should by very similar to the random, and even the approximate addition scheme. The initial $\lambda(x), \rho(x)$ have only a single term. For $d << N$, the probability of multi-edges is very low, and so the first check node will give rise to a predictable λ. The situation becomes more for later check nodes, and for a general irregular graph. Concentration-like results must be proved [1].

Simulations were run adding a series of check nodes (50 check nodes of degree 6) to a $(3, 6)$ regular-LDPC code of length $N = 1000$, by randomly connecting the check nodes to variable nodes. This random addition scheme was compared with the average, and the approximate addition scheme. Density Evolution for Gallager B algorithm was run (for 10 iterations) on each of the 3 schemes. We chose to add degree-6 check nodes in order to keep $\rho(x)$ a constant. Optimizations in choosing check node degrees might be possible [2].

Adding check nodes indeed improve the performance of the LDPC-code. The average (expected) degree distribution differs from the random allocation, by a maximum of 10^{-3} in each term, in both the cases. The performance of the average and the random addition schemes under Gallager B (density evolution) were almost identical. The approximate scheme differed slightly for the $N = 1000$ case, but was much closer for the $N = 4000$ case.

[1] ToDo - Prove concentration ?

[2] Speculation

IV. Achieving Scalable Performance

We spoke about the effects of adding a check node in Section III-A. Intuitively, adding check nodes seems to improve the performance of the LDPC code, on average. Adding check nodes can decrease the girth of the graph, but the extra information flow seems to give the (density evolution) performance quite a boost. In this section, we harness this idea to change the performance of the LDPC code as desired, by adding more check nodes. We seek to use a single parameter to change the performance of the LDPC code in stages, and show how the energy requirement and rate change at each stage.

Adding check nodes decreases the rate of the LDPC code. One way to adjust for this would be to over-design the base code (rate much higher than required and high girth) and keep adding check nodes until we hit the limit on acceptable rate.

As long as we consider existing LDPC codes with iterative decoding schemes, performance and energy seem to be at odds. Under this framework, better performance comes the price of larger codes, more wires, and more energy. The schemes we propose here are ways for achieving flexible performance, and we elaborate on the trade-offs involved in the scheme.

A. Single decoder implementation

The basic idea used is, adding check nodes in stages to achieve differing levels of performance. Say we add c-check nodes in each stage to a (λ, ρ) code of length N. At the m-th stage, a net total of mc-check nodes would have been

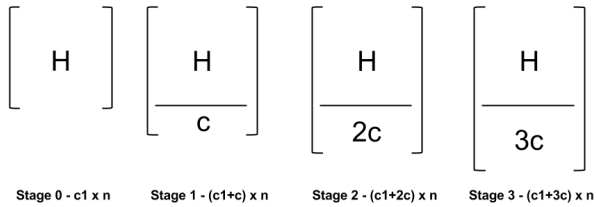


Fig. 3: Parity Check Matrix of a Single Local decoder with c nodes added in each stage, i.e mc nodes are added to the base decoder at Stage m

added. We can measure the average of this using the methods described in Section III-B. Figure 3 shows the change in the parity check matrices used in each Stage.

This was tested using an initially regular-$(3, 6)$-LDPC code of length $N = 2000$. In stage $m = 1$, 50 check nodes (of degree 6) were added. In stage $m = 2$, 50 more check nodes were added to the above configuration, giving 100 nodes over the initial code. In stage $m = 3$, 50 more were added, giving 150, and 50 more were added in stage 4. Density evolution on Gallager-B was run to 10-iterations.

Figures 4 and 5 show the stage-wise performance of the code after adding check nodes. We can see from 5 that the probability of error after the 10-iterations decays almost exponentially with m. This means that adding check nodes appears to be exponentially beneficial in terms of performance. The initial code has a rate 0.5, and the final code has a rate 0.4.
[3]

Every time a check node is added to the decoder, we are adding extra wires connecting the nodes to the existing circuit. So, in every stage, we end up using more wires, more bit-meters, and subsequently, more energy. There is a trade-off between the performance and the energy consumption of the circuit, which is broadly what we want to bring out in this paper. We used an average analysis, and basic scheme for scaling performance to show this idea, though more sophisticated schemes might exist.

B. Energy Used by the Single Decoder Method

To get an estimate of the extra energy used in each stage, we use ideas from [1] and the *unpublished Journal article* [4]. We know that for a circuit with N-nodes, the number of bit-meter scales roughly as $\mathcal{O}(N^2)$.

We know that the initial decoder circuit has N variable nodes, and M-edges. So the total number of nodes in the decoder circuit it. In Stage-m, mc check nodes are added ($c = 50$ in the example above). So, each of these check nodes has a certain degree. Assume for now that all the check nodes added have the same degree d_c.

$$N_{total} = N\left(1 + \frac{\int \rho(x)dx}{\int \lambda(x)dx}\right)$$

[3]Possibility of an analytical version of the same result ? Hard to prove for any p_0 below threshold

[4]Karthik Ganesan, Pulkit Grover

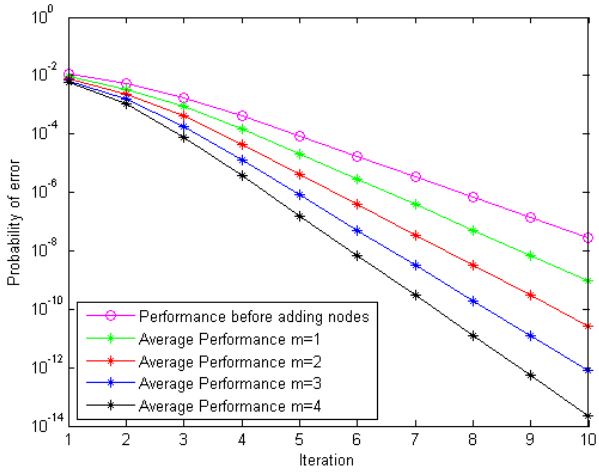


Fig. 4: Stage-wise comparison of Density Evolution, obtained by adding 50 nodes at each stage to the regular-$(3, 6)$-LDPC code of length $N = 2000$. Density evolution was run to 10 iterations, with $p_0 = 0.02$.

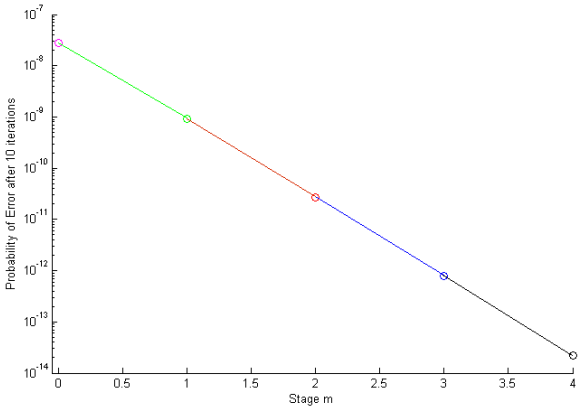


Fig. 5: Exponential decay of the probability of error with m (after 10 iterations)

For the generic upper bound on energy for the circuit, we proceed as follows. In Stage-m, the total number of nodes in the circuit would be $N_{total} + mc$. The total number of edges is $M + mcd_c$. So, the upper bound on number of bit-meters is

$$BM_{total} = \mathcal{O}((N + mc)^2 N_{iter}) \tag{13}$$

Now we take a specific circuit drawing, to aid comparison. Assuming we are able to lay arrange all nodes and wires in a circle, the radius r of the smallest encompassing circle is given by

$$\begin{aligned} \pi r^2 &\geq c' N_{total} = c'' N \\ r &= \Omega(\sqrt{N}) \end{aligned} \tag{14}$$

Also, the upper bound on r can be obtained by arranging all the nodes in a line. If each node occupies a constant area, the diameter of the circle would be propertional to the number of nodes in the circuit. That is, $r = \mathcal{O}(N)$.

Now, we are interested in the extra energy consumption in the various stages. Assume that in each stage we place

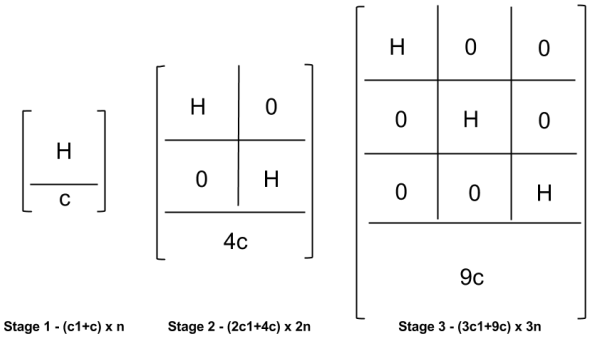


Fig. 6: Combining smaller decoders to create a global decoder. m^2c nodes are added to the global decoder at Stage m

the nodes outside the local decoder. Assume for now that the nodes occupy small enough area, and the number of nodes added is small enough that they can fit on the circumference of the circle. The maximum wire length required to connect a single node to the local decoder is the diameter of the circle $2r$. So, an upper bound on the number of extra bit-meters needed would be

$$BM_{extra} \leq mcd_c(2r)N_{iter}$$
$$BM_{extra} = \mathcal{O}(mN_{total}N_{iter}) \quad (15)$$

The number extra bit-meters required for the circuit increases at-most linearly in m (assuming $mc << N$). The exact change depends on how the circuit is laid out.

C. Global Decoder Method

The global decoder method is a way to (potentially) avoid the girth reduction of the local decoder described in the previous section, while trying to maintain its performance. The idea is to increase the overall length of the codeword by placing the H matrices serially (as shown in 6). We then add random check noses to the serial combination, for the scaled performance.

Since the length of the codeword is higher, the number of check nodes needed to get the same performance scaling as in IV-A is higher. In particular, we need to add m^2c check nodes to get the close to the same performance benefit (by density evolution). We can understand this by considering one particular case where we can separate each of the mc check nodes to each local decoder (H matrix) (as is the case in IV-A).

Since there are m local decoders, we need m^2c nodes. The performance of any specific method of adding check nodes concentrates [5] around the average case for $d << N$. So, we consider the case of adding m^2c check nodes. Simulations tell us the performance gain is similar to that seen in Fig. 4 and Fig. 5. We still add a constant d_c edges to each check node (keeping the check node degree distribution a constant), but these edges are now dispersed among all the local decoders' variable nodes. As a result, the girth of the graph is less likely to decrease as compared to the local decoder case [6].

[5] Yet to be proved

[6] Untested

D. Energy Used by the Global Decoder Method, and Comparisons

We first compute the general bounds on energy used by the global decoder method, and compare it to the local decoder method. Say out initial H matrix had $c1$-check nodes, and N variable nodes, and the total number of nodes being $N_{total} = N + c1$. In Stage-m, we have m of these decoders, and m^2c extra nodes. So the total number of nodes in the decoder graph is $N_{total,m} = mN_{total} + m^2c$. Since the upper bound on the total number of bit-meters is $\mathcal{O}((N_{total,m})^2)$, we get

$$BM_{total}^{global} = \mathcal{O}((mN_{total} + m^2c)^2 N_{iter}^{global})$$
$$= \mathcal{O}(m^2(N_{total} + mc)^2 N_{iter}^{global}) \quad (16)$$

To compare this with the case of m-local decoders, we take (13), and repeat for m-independent local decoders, giving us,

$$BM_{total}^{local} = \mathcal{O}(m(N_{total} + mc)^2 N_{iter}^{local}) \quad (17)$$

which is a while order of m-lower (better) than the global decoder case. In the absence of girth benefit, local decoders consume far less energy asymptotically. Tests with a specific drawing also show that the wire-length is larger in the global decoder as compared to the local decoder scheme.

V. Conclusion

We developed the mathematical framework for expected change in the degree distribution of irregular graphs after node addition. We used this to propose two schemes for achieving flexible performance of LDPC codes. We also showed how the increased performance comes at the cost of more energy, gave bounds on the extra energy required for each scheme, and compared the two.

Appendix A
Approximate Degree Distribution

This formula is currently not used, though is probably necessary for the analysis of performance. Later, either the proof will be described here, or the formula will be removed.

References

[1] Pulkit Grover, Information-friction and its impact on minimum energy per communicated bit, in *IEEE International Symposium on Information Theory Proceedings (ISIT), 2013, Pages 2513 - 2517*

[2] Gallager, R. G., "Low Density Parity Check Codes", Transactions of the IRE Professional Group on Information Theory, Vol. IT-8, January 1962, pp. 2l-28.

[3] R. M. Tanner, A recursive approach to low complexity codes, IEEE Trans. Inform. Theory, vol. IT-27, pp. 533547, Sept. 1981.

[4] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Efficient erasure correcting codes. IEEE Transactions on Information Theory, 47(2):569584, 2001

[5] Venkatesan Guruswami, "Iterative Decoding of Low-Density Parity Check Codes (A Survey)", arXiv: http://arxiv.org/abs/cs/0610022

[6] Amin Shokrollahi, "LDPC Codes: An Introduction", Coding, Cryptography and Combinatorics

[7] Thomas J. Richardson and Rdiger L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding", IEEE Transactions on Information Theory, Vol. 47, NO. 2, February 2001

[8] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density parity- check codes using irregular graphs. IEEE Transactions on Information Theory, 47(2):585598, 2001.