LDPC Codes – a brief Tutorial

Bernhard M.J. Leiner, Stud.ID.: 53418L
bleiner@gmail.com

April 8, 2005

1 Introduction

Low-density parity-check (LDPC) codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few 1's in comparison to the amount of 0's. Their main advantage is that they provide a performance which is very close to the capacity for a lot of different channels and linear time complex algorithms for decoding. Furthermore are they suited for implementations that make heavy use of parallelism.

LDPC

They were first introduced by Gallager in his PhD thesis in 1960. But due to the computational effort in implementing coder and encoder for such codes and the introduction of Reed-Solomon codes, they were mostly ignored until about ten years ago.

GALLAGER, 1960

1.1 Representations for LDPC codes

Basically there are two different possibilities to represent LDPC codes. Like all linear block codes they can be described via matrices. The second possibility is a graphical representation.

Matrix Representation

Lets look at an example for a low-density parity-check matrix first. The matrix defined in equation (1) is a parity check matrix with dimension $n \times m$ for a $(8,4)$ code.

We can now define two numbers describing these matrix. w_r for the number of 1's in each row and w_c for the columns. For a matrix to be called *low-density* the two conditions $w_c \ll n$ and $w_r \ll m$ must

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

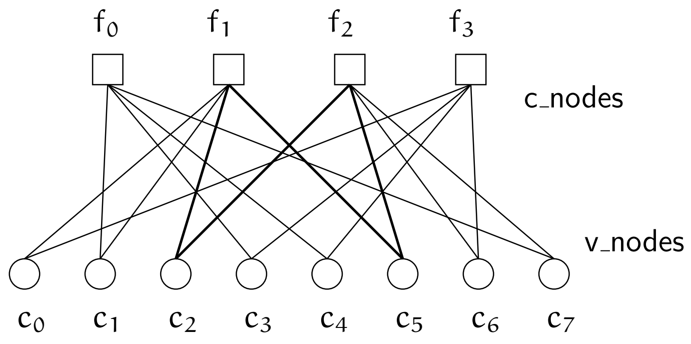

Figure 1: Tanner graph corresponding to the parity check matrix in equation (1). The marked path $c_2 \to f_1 \to c_5 \to f_2 \to c_2$ is an example for a short cycle. Those should usually be avoided since they are bad for decoding performance.

be satisfied. In order to do this, the parity check matrix should usually be very large, so the example matrix can't be really called low-density.

Graphical Representation

Tanner introduced an effective graphical representation for LDPC codes. Not only provide these graphs a complete representation of the code, they also help to describe the decoding algorithm as explained later on in this tutorial.

TANNER GRAPH, 1981

Tanner graphs are *bipartite graphs*. That means that the nodes of the graph are separated into two distinctive sets and edges are only connecting nodes of two different types. The two types of nodes in a Tanner graph are called *variable nodes* (*v-nodes*) and *check nodes* (*c-nodes*).

V-NODES
C-NODES

Figure 1.1 is an example for such a Tanner graph and represents the same code as the matrix in 1. The creation of such a graph is rather straight forward. It consists of m check nodes (the number of parity bits) and n variable nodes (the number of bits in a codeword). Check node f_i is connected to variable node c_j if the element h_{ij} of $\mathbf{H}$ is a 1.

1.2 Regular and irregular LDPC codes

A LDPC code is called *regular* if w_c is constant for every column and $w_r = w_c \cdot (n/m)$ is also constant for every row. The example matrix from equation (1) is regular with $w_c = 2$ and $w_r = 4$. It's also possible to see the regularity of this code while looking at the graphical representation. There is the same number of incoming edges for every v-node and also for all the c-nodes. REGULAR

If $\mathbf{H}$ is low density but the numbers of 1's in each row or column aren't constant the code is called a *irregular* LDPC code. IRREGULAR

1.3 Constructing LDPC codes

Several different algorithms exists to construct suitable LDPC codes. Gallager himself introduced one. Furthermore MacKay proposed one to semi-randomly generate sparse parity check matrices. This is quite interesting since it indicates that constructing good performing LDPC codes is not a hard problem. In fact, completely randomly chosen codes are good with a high probability. The problem that will arise, is that the encoding complexity of such codes is usually rather high. MacKay

2 Performance & Complexity

Before describing decoding algorithms in section 3, I would like to explain why all this effort is needed. The feature of LDPC codes to perform near the Shannon limit[1] of a channel exists only for large block lengths. For example there have been simulations that perform within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} with an block length of 10^7. An interesting fact is that those high performance codes are irregular. Shannon limit

The large block length results also in large parity-check and generator matrices. The complexity of multiplying a codeword with a matrix depends on the amount of 1's in the matrix. If we put the sparse matrix $\mathbf{H}$ in the form $[\mathbf{P}^T \; \mathbf{I}]$ via Gaussian elimination the generator matrix $\mathbf{G}$ can be calculated as $\mathbf{G} = [\mathbf{I} \; \mathbf{P}]$. The sub-matrix $\mathbf{P}$ is generally not sparse so that the encoding complexity will be quite high.

[1]Shannon proofed that reliable communication over an unreliable channel is only possible with code rates above a certain limit – the channel capacity.

Since the complexity grows in $O(n^2)$ even sparse matrices don't result in a good performance if the block length gets very high. So iterative decoding (and encoding) algorithms are used. Those algorithms perform local calculations and pass those local results via messages. This step is typically repeated several times.

The term "local calculations" already indicates that a divide and conquere strategy, which separates a complex problem into manageable sub-problems, is realized. A sparse parity check matrix now helps this algorithms in several ways. First it helps to keep both the local calculations simple and also reduces the complexity of combining the sub-problems by reducing the number of needed messages to exchange all the information. Furthermore it was observed that iterative decoding algorithms of sparse codes perform very close to the optimal maximum likelihood decoder.

DIVIDE AND CONQUERE

3 Decoding LDPC codes

The algorithm used to decode LDPC codes was discovered independently several times and as a matter of fact comes under different names. The most common ones are the *belief propagation algorithm* , the *message passing algorithm* and the *sum-product algorithm*.

BPA
MPA
SPA

In order to explain this algorithm, a very simple variant which works with hard decision, will be introduced first. Later on the algorithm will be extended to work with soft decision which generally leads to better decoding results. Only binary symmetric channels will be considered.

3.1 Hard-decision decoding

The algorithm will be explained on the basis of the example code already introduced in equation 1 and figure 1.1. An error free received codeword would be e.g. $c = [1\ 0\ 0\ 1\ 0\ 1\ 0\ 1]$. Let's suppose that we have a BHC channel and the received the codeword with one error – bit c_1 flipped to 1.

$c_i \rightarrow f_j$

1. In the first step all v-nodes c_i send a "message" to their (always 2 in our example) c-nodes f_j containing the bit they believe to be the correct one for them. At this stage the only information a v-node c_i has, is the corresponding received i-th bit of c, y_i. That means for example, that c_0 sends a message containing 1 to f_1 and f_3, node c_1 sends messages containing y_1 (1) to f_0 and f_1, and so on.

c-node	received/sent				
f_0	received:	$c_1 \rightarrow 1$	$c_3 \rightarrow 1$	$c_4 \rightarrow 0$	$c_7 \rightarrow 1$
	sent:	$0 \rightarrow c_1$	$0 \rightarrow c_3$	$1 \rightarrow c_4$	$0 \rightarrow c_7$
f_1	received:	$c_0 \rightarrow 1$	$c_1 \rightarrow 1$	$c_2 \rightarrow 0$	$c_5 \rightarrow 1$
	sent:	$0 \rightarrow c_0$	$0 \rightarrow c_1$	$1 \rightarrow c_2$	$0 \rightarrow c_5$
f_2	received:	$c_2 \rightarrow 0$	$c_5 \rightarrow 1$	$c_6 \rightarrow 0$	$c_7 \rightarrow 1$
	sent:	$0 \rightarrow c_2$	$1 \rightarrow c_5$	$0 \rightarrow c_6$	$1 \rightarrow c_7$
f_3	received:	$c_0 \rightarrow 1$	$c_3 \rightarrow 1$	$c_4 \rightarrow 0$	$c_6 \rightarrow 0$
	sent:	$1 \rightarrow c_0$	$1 \rightarrow c_3$	$0 \rightarrow c_4$	$0 \rightarrow c_6$

Table 1: overview over messages received and sent by the c-nodes in step 2 of the message passing algorithm

2. In the second step every check nodes f_j calculate a response to every connected variable node. The response message contains the bit that f_j believes to be the correct one for this v-node c_i assuming that the other v-nodes connected to f_j are correct. In other words: If you look at the example, every c-node f_j is connected to 4 v-nodes. So a c-node f_j looks at the message received from three v-nodes and calculates the bit that the fourth v-node should have in order to fulfill the parity check equation. Table 2 gives an overview about this step. $f_j \rightarrow c_i$

   Important is, that this might also be the point at which the decoding algorithm terminates. This will be the case if all check equations are fulfilled. We will later see that the whole algorithm contains a loop, so an other possibility to stop would be a threshold for the amount of loops.

3. Next phase: the v-nodes receive the messages from the check nodes and use this additional information to decide if their originally received bit is OK. A simple way to do this is a majority vote. When coming back to our example that means, that each v-node has three sources of information concerning its bit. The original bit received and two suggestions from the check nodes. Table 3 illustrates this step. Now the v-nodes can send another message with their (hard) decision for the correct value to the check nodes. $c_i \rightarrow f_j$

4. Go to step 2. LOOP

In our example, the second execution of step 2 would terminate the decoding process since c_1 has voted for 0 in the last step. This corrects

v-node	y_i received	messages from check nodes		decision
c_0	1	$f_1 \rightarrow 0$	$f_3 \rightarrow 1$	1
c_1	1	$f_0 \rightarrow 0$	$f_1 \rightarrow 0$	0
c_2	0	$f_1 \rightarrow 1$	$f_2 \rightarrow 0$	0
c_3	1	$f_0 \rightarrow 0$	$f_3 \rightarrow 1$	1
c_4	0	$f_0 \rightarrow 1$	$f_3 \rightarrow 0$	0
c_5	1	$f_1 \rightarrow 0$	$f_2 \rightarrow 1$	1
c_6	0	$f_2 \rightarrow 0$	$f_3 \rightarrow 0$	0
c_7	1	$f_0 \rightarrow 1$	$f_2 \rightarrow 1$	1

Table 2: Step 3 of the described decoding algorithm. The v-nodes use the answer messages from the c-nodes to perform a majority vote on the bit value.

the transmission error and all check equations are now satisfied.

3.2 Soft-decision decoding

The above description of hard-decision decoding was mainly for educational purpose to get an overview about the idea. Soft-decision decoding of LDPC codes, which is based on the concept of belief propagation, yields in a better decoding performance and is therefore the prefered method. The underlying idea is exactly the same as in hard decision decoding. Before presenting the algorithm lets introduce some notations:

BELIEF PROPAGATION

- $P_i = Pr(c_i = 1|y_i)$
- q_{ij} is a message sent by the variable node c_i to the check node f_j. Every message contains always the pair $q_{ij}(0)$ and $q_{ij}(1)$ which stands for the amount of belief that y_i is a "0" or a "1".
- r_{ji} is a message sent by the check node f_j to the variable node c_i. Again there is a $r_{ji}(0)$ and $r_{ji}(1)$ that indicates the (current) amount of believe in that y_i is a "0" or a "1".

The step numbers in the following description correspond to the hard decision case.

1. All variable nodes send their q_{ij} messages. Since no other information is avaiable at this step, $q_{ij}(1) = P_i$ and $q_{ij}(0) = 1 - P_i$.

$c_i \rightarrow f_j$

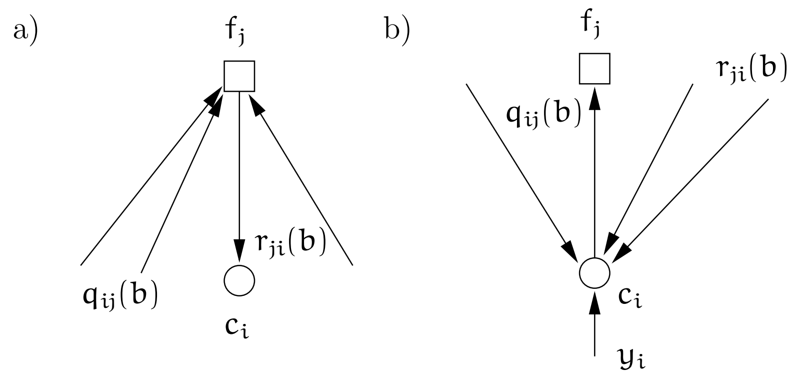

Figure 2: a) illustrates the calculation of $r_{ji}(b)$ and b) $q_{ij}(b)$

2. The check nodes calculate their response messages r_{ji}:[2] $f_j \rightarrow c_i$

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_j \backslash i} (1 - 2q_{i'j}(1)) \tag{3}$$

   and

$$r_{ji}(1) = 1 - r_{ji}(0) \tag{4}$$

   So they calculate the probability that there is an even number of 1's amoung the variable nodes except c_i (this is exactly what $V_j \backslash i$ means). This propability is equal to the probability $r_{ji}(0)$ that c_i is a 0. This step and the information used to calculate the responses is illustrated in figure 2.

3. The variable nodes update their response messages to the check nodes. This is done according to the following equations, $c_i \rightarrow f_j$

$$q_{ij}(0) = K_{ij}\,(1 - P_i) \prod_{j' \in C_i \backslash j} r_{j'i}(0) \tag{5}$$

$$q_{ij}(1) = K_{ij} P_i \prod_{j' \in C_i \backslash j} r_{j'i}(1) \tag{6}$$

[2]Equation 3 uses the following result from Gallager: for a sequence of M independent binary digits a_i with an propability of p_i for $a_i = 1$, the probability that the whole sequence contains an *even* number of 1's is

$$\frac{1}{2} + \frac{1}{2} \prod_{i=1}^{M} (1 - 2p_i) \tag{2}$$

whereby the Konstants K_{ij} are chosen in a way to ensure that $q_{ij}(0) + q_{ij}(1) = 1$. $C_i \backslash j$ now means all check nodes except f_j. Again figure 2 illustrates the calculation in this step.

At this point the v-nodes also update their current estimation $\hat{c}_i$ of their variable c_i. This is done by calculating the propabilities for 0 and 1 and voting for the bigger one. The used equations

$$Q_i(0) = K_i (1 - P_i) \prod_{j \in C_i} r_{ji}(0) \tag{7}$$

and

$$Q_i(1) = K_i P_i \prod_{j \in C_i} r_{ji}(1) \tag{8}$$

are quite similar to the ones to compute $q_{ij}(b)$ but now the information from every c-node is used.

$$\hat{c}_i = \begin{cases} 1 & \text{if } Q_i(1) > Q_i(0), \\ 0 & \text{else} \end{cases} \tag{9}$$

If the current estimated codeword fufills now the parity check equations the algorithm terminates. Otherwise termination is ensured through a maximum number of iterations.

4. Go to step 2. LOOP

The explained soft decision decoding algorithm is a very simple variant, suited for BSC channels and could be modified for performance improvements. Beside performance issues there are nummerical stability problems due to the many multiplications of probabilities. The results will come very close to zero for large block lenghts. To prevent this, it is possible to change into the log-domain and doing additions instead of multiplications. The result is a more stable algorithm that even has performance advantages since additions are less costly. LOG-DOMAIN

4 Encoding

The sharped eyed reader will have noticed that the above decoding algorithm does in fact only error correction. This would be enough for a traditional systematic block code since the code word would consist of the message bits and some parity check bits. So far nowhere was mentioned that it's possible to directly see the original message bits in a LDPC encoded message. Luckily it is.

Encoding LDPC codes is roughly done like that: Choose certain variable nodes to place the message bits on. And in the second step calculate the missing values of the other nodes. An obvious solution for that would be to solve the parity check equations. This would contain operations involving the whole parity-check matrix and the complexity would be again quadratic in the block length. In practice however, more clever methods are used to ensure that encoding can be done in much shorter time. Those methods can use again the spareness of the parity-check matrix or dictate a certain structure[3] for the Tanner graph.

5 Summary

Low-density-parity-check codes have been studied a lot in the last years and huge progresses have been made in the understanding and ability to design iterative coding systems. The iterative decoding approach is already used in turbo codes but the structure of LDPC codes give even better results. In many cases they allow a higher code rate and also a lower error floor rate. Furthermore they make it possible to implement parallelizable decoders. The main disadvantes are that encoders are somehow more complex and that the code lenght has to be rather long to yield good results.

For more information – and there are a lot of things which haven't been mentioned in this tutorial – I can recommend the following web sites as a start:

`http://www.csee.wvu.edu/wcrl/ldpc.htm`
: A collection of links to sites about LDPC codes and a list of papers about the topic.

`http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html`
: The homepage of MacKay. Very intersting are the *Pictorial demonstration of iterative decoding*

[3]It was already mentioned in section 1.3 that randomly generated LDPC codes results in high encoding complexity.