Implementation of LDPC codes

IDP(EE2025)
June 30, 2020

SHAIK MASTAN VALI
J PRABHATH LAKSHMINARAYANA
K SRIKANTH

Mentor: Dr. Shashank Vatedka
INDIAN INSTITUTE OF TECHNOLOGY, HYDERABAD

Table of contents

1. Low-density parity-check(LDPC) codes are codes specified by a parity check matrix H containing mostly 0's and only a small number of 1's.
2. A regular (n, w_c, w_r) LDPC code of a code of block-length n with a m$\times$ n parity check matrix where each column contains a small fixed number , $w_c \geq 3$, of 1's and each row contains a small fixed number, $w_r \geq w_c$
3. in other words
- Each parity check constraint involves w_r code bits, and each code bit is involved in w_c constraints.
- Low-density implies that $w_c << m$ and $w_r << n$
- Number of ones in the parity check matrix H = w_c n = w_r m
- $m \geq n - k \quad \Rightarrow \quad$ R = k/n $\geq 1 - (w_c/w_r)$. and thus $w_c < W_r$.

Regular LDPC

Regular Low-density parity check code:

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Example of Regular Low-density code matrix; n=20, w_c = 3, w_r = 4

Tanner Graphs

1. A bipartite graph is one in which the nodes can be partitioned into two classes, and no edge can connect nodes from the same class.
2. A tanner graph for an LDPC code is an bipartite graph such that:
   - One class of the nodes is the "Variable nodes" corresponding to ’n’ bits in the code word.
   - Second class of nods is "check nodes" corresponding to ’m’ parity check equations.
   - An edge connects a variable node to the check node if and only if that particular bit is included in the parity check equation.

Gallagher's Construction for regular (n, w_c , w_r) code

- Let, n be the transmitted block-length of an information sequence of length k. m is the number of parity check equations.
- Construct a m $\times$ n matrix with w_c 1's per column and w_r 1's per row.(An (n, w_c , w_r)code).
- Divide a m$\times$n matrix into W_c, m/W_c $\times$n sub-matrices, each containing a single 1 in each column.
- The first of these sub-matrices contains all's in descending order. i.e the i^{th} row contains 's in column(i - 1).w_r + 1 to i.w_r.
- The order sub-matrices are merely column permutations of the first sub-matrix.

Decoder for an AWGN channel

In the Tanner graph, the bit nodes are called repetition nodes and the check nodes are called zero-sum nodes. The incoming information at each bit node corresponds to that single variable only, which is the same case as a repetition code and the . The incoming information at each check node corresponds to the parity check constraint, which is also called as zero-sum constraint.

So, we split the belief propagation decoder into two parts - SISO (soft-input-soft output) decoder for a repetition code and SISO decoder for SPC (Single Parity Check) code.

Consider a (3, 1) repetition codeblock (c_1, c_2, c_3).
Let L_i = "belief" that $c_i = 0$.
Output of the decoder: intrinsic + extrinsic

Intrinsic belief (Log-likelihood ratio):

$$Pr(c_1 = 0|r_1) = \frac{f(r_1|c_1 = 0)Pr(c_1 = 0)}{f(r_1)}$$

$$Pr(c_1 = 1|r_1) = \frac{f(r_1|c_1 = 1)Pr(c_1 = 1)}{f(r_1)}$$

We are using BPSK modulation scheme. If $c_1 = 0$, symbol = +1.
$\Rightarrow r_1 = 1 + \mathcal{N}(0, \sigma^2)$
Similarly, if $c_1 = 1$, symbol = -1.
$\Rightarrow r_1 = -1 + \mathcal{N}(0, \sigma^2)$

SISO decoder for Repetition code (contd...)

$$\text{Likelihood ratio : } \frac{Pr(c_1 = 0|r_1)}{Pr(c_1 = 1|r_1)} = \frac{f(r_1|c_1 = 0)}{f(r_1|c_1 = 1)}$$

$$\Rightarrow \frac{Pr(c_1 = 0|r_1)}{Pr(c_1 = 1|r_1)} = \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{(r_1-1)^2}{2\sigma^2})}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{(r_1+1)^2}{2\sigma^2})} = \exp(\frac{2r_1}{\sigma^2})$$

$$\text{Log-Likelihood ratio : } log\left(\frac{Pr(c_1 = 0|r_1)}{Pr(c_1 = 1|r_1)}\right) = \frac{2r_1}{\sigma^2}$$

$$= r_1 \times constant$$

Generally, the constant is ignored. This LLR is called intrinsic LLR (or input LLR or channel LLR)

SISO decoder for Repetition code (contd...)

Output LLR:

$$L_i = log\left(\frac{Pr(c_i = 0|r_1, r_2, r_3)}{Pr(c_i = 1|r_1, r_2, r_3)}\right)$$

$$L_1 : \frac{Pr(c_1 = 0|r_1, r_2, r_3)}{Pr(c_1 = 1|r_1, r_2, r_3)} = \frac{f(r_1r_2r_3|c_1 = 0)}{f(r_1r_2r_3|c_1 = 1)}$$

Suppose $c_1 = 0 \Rightarrow$ symbol vector $= [+1, +1, +1]$

$$r_1 = 1 + \mathcal{N}_1(0, \sigma^2)$$
$$r_2 = 1 + \mathcal{N}_2(0, \sigma^2)$$
$$r_3 = 1 + \mathcal{N}_3(0, \sigma^2),$$

where $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ are i.i.d.

$$\frac{Pr(c_1 = 0|r_1, r_2, r_3)}{Pr(c_1 = 1|r_1, r_2, r_3)} = \frac{\exp(\frac{(r_1-1)^2}{2\sigma^2})\exp(\frac{(r_2-1)^2}{2\sigma^2})\exp(\frac{(r_3-1)^2}{2\sigma^2})}{\exp(\frac{(r_1+1)^2}{2\sigma^2})\exp(\frac{(r_2+1)^2}{2\sigma^2})\exp(\frac{(r_3+1)^2}{2\sigma^2})}$$

$$= \exp(\frac{2(r_1 + r_2 + r_3)}{\sigma^2})$$

$$\Rightarrow L_1 = \frac{2(r_1 + r_2 + r_3)}{\sigma^2}$$

$$\Rightarrow L_1 \propto (r_1 + r_2 + r_3)$$

r_1 : Intrinsic belief, $r_2 + r_3$: Extrinsic belief

SISO decoder for an SPC code

Consider a general (n, n-1) SPC code. If $\mathbf{m} = [m_1, m_2, ..., m_{n-1}]$, then the codeword $\mathbf{c} = [c_1, c_2, ..., c_{n-1}, p]$, where $p = m_1 \oplus m_2 \oplus ..... \oplus m_{n-1}$.

In an SPC codeword, the no. of 1's is even, due to the parity-check condition $c_1 \oplus c_2 ..... \oplus c_n = 0$.

Suppose n = 3. Then, we have $c_1 = c_2 \oplus c_3$.

Given $p_2 = Pr(c_2 = 0)$ and $p_3 = Pr(c_3 = 0)$, we need to find $p_1 = Pr(c_1 = 0 | r_2, r_3)$.

Truth table:

c_2	c_3	c_1
0	0	0
1	1	0
0	1	1
1	0	1

From the truth table,

$$p_1 = p_2 p_3 + (1 - p_2)(1 - p_3)$$
$$1 - p_1 = p_2(1 - p_3) + (1 - p_2)p_3$$

SISO decoder for an SPC code (contd...)

$$p_1 - (1 - p_1) = p_2(p_3 - (1 - p_3)) + (1 - p_2)((1 - p_3) - p_3)$$

$$\frac{2p_1 - 1}{p_1 + 1 - p_1} = \frac{2p_2 - 1}{p_2 + 1 - p_2} \frac{2p_3 - 1}{p_3 + 1 - p_3}$$

$$\Rightarrow \frac{1 - \frac{(1-p_1)}{p_1}}{1 + \frac{(1-p_1)}{p_1}} = \frac{1 - \frac{(1-p_2)}{p_2}}{1 + \frac{(1-p_2)}{p_2}} \frac{1 - \frac{(1-p_3)}{p_3}}{1 + \frac{(1-p_3)}{p_3}}$$

$$\frac{1 - e^{-l_{ext,1}}}{1 + e^{-l_{ext,1}}} = \frac{1 - e^{-l_2}}{1 + e^{-l_2}} \frac{1 - e^{-l_3}}{1 + e^{-l_3}},$$

$$\text{where } l_i = log\left(\frac{Pr(c_i = 0|r_i)}{Pr(c_i = 1|r_i)}\right) = \frac{p_i}{1 - p_i}$$

SISO decoder for an SPC code (contd...)

$$\frac{e^{l_{ext,1}/2} - e^{-l_{ext,1}/2}}{e^{l_{ext,1}/2} + e^{-l_{ext,1}/2}} = \frac{e^{l_2/2} - e^{-l_2/2}}{e^{l_2/2} + e^{-l_2/2}} \frac{e^{l_3/2} - e^{-l_3/2}}{e^{l_3/2} + e^{-l_3/2}}$$

$$\Rightarrow \tanh\left(\frac{l_{ext,1}}{2}\right) = \tanh\left(\frac{l_2}{2}\right) \tanh\left(\frac{l_3}{2}\right)$$

(Here, $l_{ext,1}$ is the extrinsic LLR of bit-node 1). $\because$ tanh is an odd function, we split this equation into two parts,

Sign: $sgn(l_{ext,1}) = sgn(l_2)sgn(l_3)$

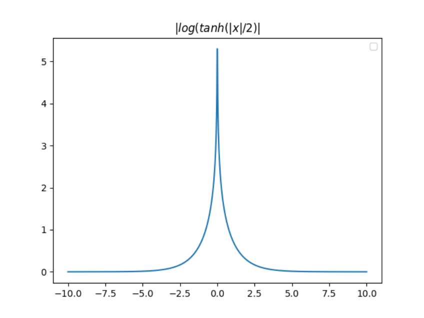
Absolute value: $\left|log\left(\tanh\left(\frac{|l_{ext,1}|}{2}\right)\right)\right| = \left|log\left(\tanh\left(\frac{|l_2|}{2}\right)\right)\right| + \left|log\left(\tanh\left(\frac{|l_3|}{2}\right)\right)\right|$

Similarly, the extrinsic LLR's for c_2 and c_3 can be derived.

SPC decoder using Min-sum approximation

We can apply an approximation to $\left|log\left(\tanh\left(\frac{|x|}{2}\right)\right)\right|$, since it is a steep function.

Let $f(x) = \left|log\left(\tanh\left(\frac{|x|}{2}\right)\right)\right|$.
$\because f(x)$ is a steep function, the higher values of x do not contribute to f(x). Hence, the dominating value is the minimum element int the set of values.



For a (3, 2) SPC code,

$$\begin{aligned} f(l_{ext,1}) &= f(l_2) + f(l_3) \\ &= f(min(|l_2|, |l_3|)) \end{aligned}$$

$$\begin{aligned} \Rightarrow |l_{ext,1}| &= f^{-1}(f(min(|l_2|, |l_3|))) \\ &= min(|l_2|, |l_3|) \end{aligned}$$

Min-sum approximation for a (n, n-1) SPC code

Generalising the previously discussed approximation,

$$|l_{ext,n}| = min(|l_1|, |l_2|, ..., |l_{n-1}|)$$

To avoid the repeated computation of sgn() and the minima, we use the folowing procedure.

$$S = sgn(l(1)l(2).......l(n))$$
$$sgn(l_{ext,i}) = sgn(l_i) \times S$$

and

$$\text{Let } m_1 = min(|l_1|, |l_2|, ..., |l_n|)$$
$$pos = argmin(|l_1|, |l_2|, ..., |l_n|)$$
$$m_2 = min(|l_1|, |l_2|, ..., |l_{pos-1}|, |l_{pos+1}|, ..., |l_n|)$$

Decoding Algorithm

- At i^{th} zero-sum node,
  - Compute the parity $S_i = sgn(m_1)sgn(m_2)...sgn(m_{w_r-1})$.
  - Compute the absolute value of LLR:

$$L_i = S_i \times \sum_{j=1}^{w_r-1} \left|\log(\tanh\left|\frac{m_i}{2}\right|)\right|$$

- At i^{th} repetition node, compute

$$m_i = r_i + \sum_{j=1}^{w_c-1} L_i$$

Note that there is a slight abuse of notation; the indices 'j' denote the set of edges incident on the particular node.

Decoding Algorithm with min-sum approximation

- At i^{th} zero-sum node,
  - Compute the parity $S_i = sgn(m_1)sgn(m_2)...sgn(m_{w_r-1})$.
  - Compute $pos = argmin(|m_1|, |m_2|, ..., |m_n|)$.

  If $i \neq pos, L_i = S_i \times |m_{pos}|$

  Else, $L_i = S_i \times min(|m_1|, |m_2|, ..., |m_{pos-1}|, |m_{pos+1}|, ..., |m_n|)$
- At i^{th} repetition node, compute

$$m_i = r_i + \sum_{j=1}^{w_c-1} L_i$$

Note that there is a slight abuse of notation; the indices 'j' denote the set of edges incident on the particular node.

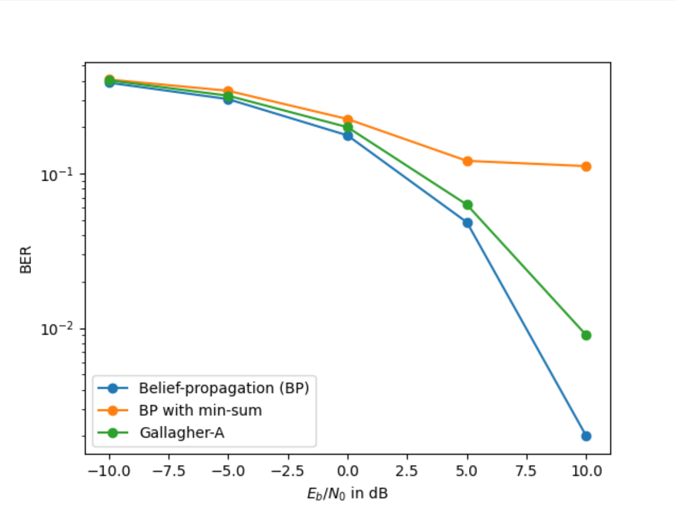
Terminal output for Belief Propagation

```
shaik-mastan@shaik-mastan-HP-Laptop-15-da1xxx:~/Signal-Processing/LDPC/AWGN$ python decode.py
Belief Propagation: ##########################################

For E_b/N_0 = -10 dB
No. of incorrectly decoded bits (without min-sum): 4272
Bit Error rate (without min-sum): 0.3883636363636363636
No. of incorrectly decoded bits (with min-sum): 4479
Bit Error rate (with min-sum): 0.4071818181818182

For E_b/N_0 = -5 dB
No. of incorrectly decoded bits (without min-sum): 3342
Bit Error rate (without min-sum): 0.3038181818181818
No. of incorrectly decoded bits (with min-sum): 3785
Bit Error rate (with min-sum): 0.3440909090909091

For E_b/N_0 = 0 dB
No. of incorrectly decoded bits (without min-sum): 1947
Bit Error rate (without min-sum): 0.177
No. of incorrectly decoded bits (with min-sum): 2484
Bit Error rate (with min-sum): 0.2258181818181818

For E_b/N_0 = 5 dB
No. of incorrectly decoded bits (without min-sum): 533
Bit Error rate (without min-sum): 0.04845454545454545
No. of incorrectly decoded bits (with min-sum): 1332
Bit Error rate (with min-sum): 0.1210909090909091

For E_b/N_0 = 10 dB
No. of incorrectly decoded bits (without min-sum): 22
Bit Error rate (without min-sum): 0.002
No. of incorrectly decoded bits (with min-sum): 1231
Bit Error rate (with min-sum): 0.1119090909090909
```

Terminal output for Gallagher-A

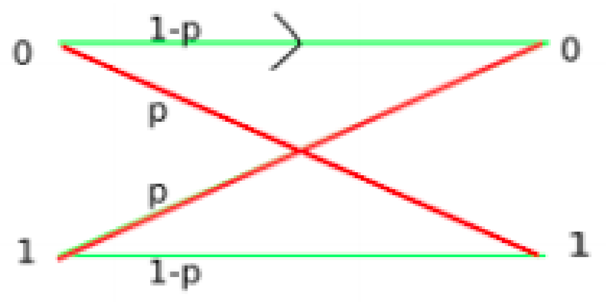
```
Gallagher-A decoding: ###########################################

For E_b/N_0 = -10 dB
No. of incorrectly decoded bits: 4423
Bit Error rate: 0.4020909090909091

For E_b/N_0 = -5 dB
No. of incorrectly decoded bits: 3518
Bit Error rate: 0.3198181818181818З

For E_b/N_0 = 0 dB
No. of incorrectly decoded bits: 2200
Bit Error rate: 0.2

For E_b/N_0 = 5 dB
No. of incorrectly decoded bits: 693
Bit Error rate: 0.063

For E_b/N_0 = 10 dB
No. of incorrectly decoded bits: 99
Bit Error rate: 0.009
```

Comparison with different algorithms



Consider a (d_v, d_c) regular LDPC code over the Binary Symmetric Channel (BSC). The BSC is completely specified by a parameter p, the crossover probability. The channel behaves as illustrated in below Figure. The correct bit is transmitted with probability 1-p and a bit flip occurs with probability p. Hence I = O = ±1. For this decoding algorithm $\mathcal{M}$ is also equal to ±1. This is called hard-decision decoding

Binary Symmetric Channel



Gallagher's A (cont...)

- $V^0(y) = y$ Variable nodes simply pass the received value as their first message

- $C^{(l)}(m_1, ..., m_{d_c-1}) = m_1 m_2 ... m_{d_c-1}$. For all l, the message from check node c to variable node v is the product of the messages received by c from all it's neighbors except v. This makes perfect sense, since if the messages c received were indeed the correct values of the code word bit, then the parity check represented by c would only be satisfied if v were the product of the messages. Note that mod 2 addition corresponds to multiplication after the identification of 0 with 1 and 1 with -1 respectively.

- $V^{(l)}(y, m_1, ..., m_{d_v-1}) = -y$ if all $m_i = -y$ and y otherwise. For all subsequent rounds, v still sends the received value, y as its message to c unless there is unanimous agreement amongst all checks except c, that v participates in.

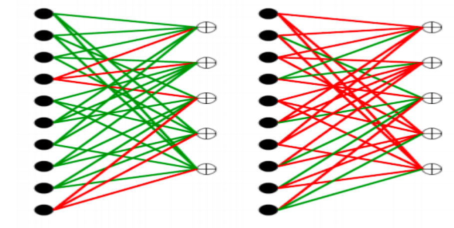
Gallagher's A (cont...)

This decoder is clearly sub-optimal since it does not even depend on the channel parameter p. Nonetheless, experimental simulations and theoretical analysis show that it is not too bad. We will analyze the asymptotic behavior of this decoder later in this section. in below Figures shows this algorithm in action. The code being used is a (3, 6) regular LDPC code. The all ones vector ++++++++++ was transmitted. Due to channel noise, the received vector is +++-+++++-. The first set of messages sent, Figure a, therefore correspond to the received values. The second set of messages, Figure b, were sent by the check nodes to the variable nodes. By 5 iterations, the algorithm converges to the vector -+---+-++- which is indeed a valid code word but not the correct one. Hence this example serves to illustrate how message passing decoders can fail. Note that a high failure rate is expected since the block length is very small.

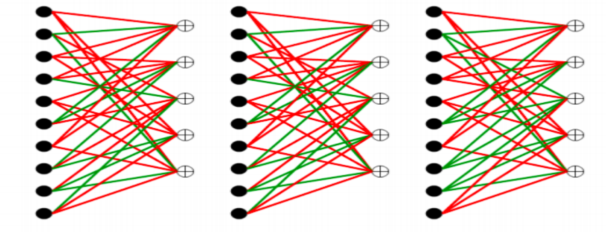
following figures shows Gallagher's algorithm A in action. R indicates the round number. The direction of arrows indicate the direction of message sent. Red corresponds to -1 and Green to 1

Gallagher’s A (cont...)

Gallagher’s algorithm A in action



(a) R0: V⟶C (b) R0: V⟵C



(c) R1: V⟶C (d) R1: V⟵C (e) R2: V⟶C

- $V^{(0)}(y) = l_i$. Here l_i is the log likelihood of the code word bit x_i conditioned on the received bit y_i. The value depends on the channel model in use. For instance, consider the binary symmetric channel with crossover probability p. Then $y_i \in$ {-1, 1}. Under the assumption of equiprobability, if the received code word was 1, then

$$l_i = \log L(x_i/y_i = 1) = \log \frac{p}{1-p}$$

otherwise

$$l_i = -\log \frac{p}{1-p}$$

- $V^{(l)}(y, m_1, ..., m_{d_v-1}) = l_i + \sum_{i=1}^{d_v-1} m_i$ The reasoning behind this is that if m_i are conditional log likelihoods of x_i, conditioned on independent random variables, then the aggregate conditional would indeed by given by this equation

Belief Propagation for BSC(cont...)

$$C^{(l)}(m_1, ..., m_{d_c-1}) = \log \frac{1 + \prod_{i=1}^{d_c-1} \tanh(m_i/2)}{1 - \prod_{i=1}^{d_c-1} \tanh(m_i/2)}$$

- If the incoming messages are independent estimates of the log likelihood of x_j , the neighbors of the check node computing the message excluding node xi, then the computed message $C^{(l)}$ correctly estimates the likelihood of x_i conditioned on the the messages and the fact that the parity check represented by the check node is satisfied.

Consequently, under the independence assumption, the belief propagation algorithm correctly performs bit wise MAP decoding.

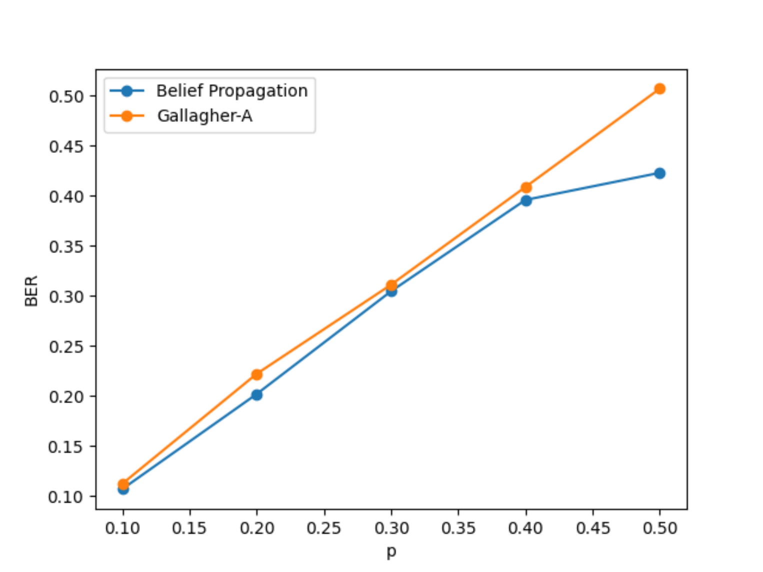
Terminal output for Belief Propagation

```
shaik-mastan@shaik-mastan-HP-Laptop-15-da1xxx:~/Signal-Processing/LDPC/BSC$ python decode.py
Belief Propagation: ##########################################

For p =  0.1
No. of incorrectly decoded bits: 1178
Bit Error rate: 0.1070909090909091

For p =  0.2
No. of incorrectly decoded bits: 2222
Bit Error rate: 0.202

For p =  0.3
No. of incorrectly decoded bits: 3352
Bit Error rate: 0.30472727272727274

For p =  0.4
No. of incorrectly decoded bits: 4356
Bit Error rate: 0.396

For p =  0.5
No. of incorrectly decoded bits: 4654
Bit Error rate: 0.4230909090909091
```

Terminal output for Gallagher-A

```
Gallagher-A: ##########################################

For p =  0.1
No. of incorrectly decoded bits: 1236
Bit Error rate: 0.11236363636363636

For p =  0.2
No. of incorrectly decoded bits: 2444
Bit Error rate: 0.22218181818181817

For p =  0.3
No. of incorrectly decoded bits: 3424
Bit Error rate: 0.31127272727272726

For p =  0.4
No. of incorrectly decoded bits: 4498
Bit Error rate: 0.4089090909090909

For p =  0.5
No. of incorrectly decoded bits: 5579
Bit Error rate: 0.5071818181818182
```

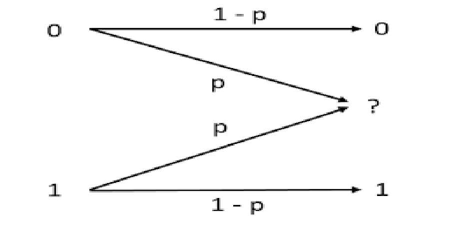
Comparison between the two decoders



Binary Erasure Channel(BEC)

The binary erasure channel (BEC) models a memoryless channel with two inputs {0, 1} andthree outputs, {0, 1, ?}, where "?" is an "erasure symbol." The probability that any transmitted bit will be received correctly is 1 - p, that it will be erased is p, and that it will be received incorrectly is zero. These transition probabilities are summarized in Figure below.

Transition probabilities of the binary erasure channel



Iterative decoding of LDPC codes on the BEC

On the binary erasure channel, the sum-product algorithm is greatly simplified, because at any time every variable corresponding to every edge in the code graph is either known perfectly (unerased) or not known at all (erased). Iterative decoding using the sum-product algorithm therefore reduces simply to the propagation of unerased variables through the code graph.

There are only two types of nodes in a normal graph of an LDPC code repetition nodes and zero-sum nodes. If all variables are either correct or erased, then the sum-product update rule for a repetition node reduces simply to:

- If any incident variable is unerased, then all other incident variables may be set equal to that variable, with complete confidence; otherwise, all incident variables remain erased.

For a zero-sum node, the sum-product update rule reduces to:

- If all but one incident variable is unerased, then the remaining incident variable may be set equal to the mod-2 sum of those inputs, with complete confidence; otherwise, variable assignments remain unchanged.

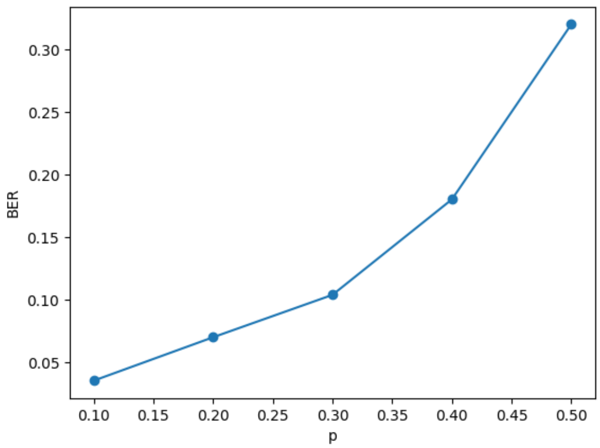
Iterative decoding of LDPC codes on the BEC(cont...)

Since all unerased variables are correct, there is no chance that these rules could produce a variable assignment that conflicts with another assignment.

Terminal output

```
shaik-mastan@shaik-mastan-HP-Laptop-15-da1xxx:~/Signal-Processing/LDPC/BEC$ python decode.py
For p = 0.1
No. of incorrectly decoded bits: 389
Bit Error rate: 0.03536363636363636

For p = 0.2
No. of incorrectly decoded bits: 771
Bit Error rate: 0.07009090909090909

For p = 0.3
No. of incorrectly decoded bits: 1144
Bit Error rate: 0.104

For p = 0.4
No. of incorrectly decoded bits: 1984
Bit Error rate: 0.18036363636363636

For p = 0.5
No. of incorrectly decoded bits: 3524
Bit Error rate: 0.32036363636363635
```

Plot



References

- Information theory, Inference and Learning Algorithms - David.J.C.McKaycv
- NPTEL
- www.stanford.edu

The End