UNIVERSITY OF TWENTE.

Information Theory and Statistics

CCSDS Low-Density Parity Check Codes - Decoding with the Sum-Product Algorithm

Group 25:

Christos Siantsis, Aristotle University of Thessaloniki
Christoforos Tsiolakis, Aristotle University of Thessaloniki
Yueming Wu, University of Twente

November 8, 2023

Introduction

The project comprises of two main components. The first component aims at devising efficient Parity Check Matrix and Generator Matrix pairs in line with the Consultative Committee for Space Data Systems (CCSDS) guidelines. These pairs are intended for utilization in spacecraft communications, where they are expected to offer high reliability even in low signal-to-noise ratio (SNR) communication channels. The second component involves implementing the Sum-Product algorithm, also known as the Belief Propagation algorithm, and presenting a simple numerical example to demonstrate its functionality. Furthermore, the Sum-Product algorithm is employed to decode a message that has been encoded using the Generator Matrix specified in the CCSDS standards.

1 LDPC Codes

1.1 General Information

Low-Density-Parity-Check (LDPC) codes were first introduced by Gallager in 1962 [1]. However, their practical use and development gained momentum only when computational hardware became powerful enough for efficient decoding [3]. LDPC Codes are linear codes. This means that for the encoding of a word t that needs to be transmitted, a generator matrix G with k linearly independent rows is multiplied with t. As a result the encoded codeword $C(t)$ will be

$$C(t) = tG \tag{1}$$

The generator matrix G is said to be in standard form when

$$G = [I_k|P] \tag{2}$$

where I_k is the $k \times k$ identity matrix and P is the $k \times n-k$ matrix that contains the redundant parity check bits, n is the codeword length and k is is the length of the vector t. The parity check matrix H is as a result

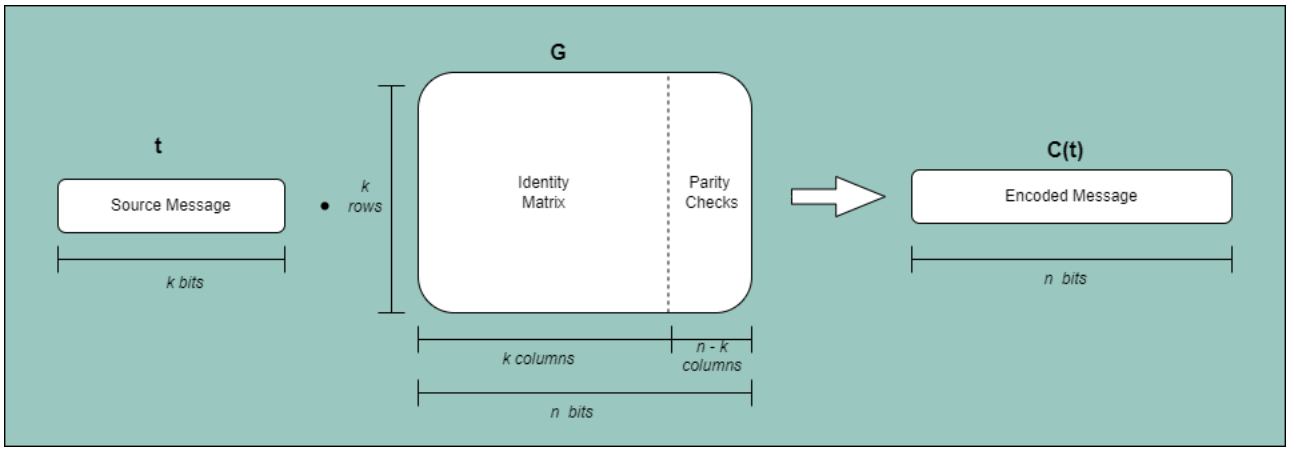

Figure 1: Linear Code Encoding with standard form Generator matrix

the matrix that in its $n-k$ rows contains the coefficients of the parity checks.

$$H = [P^T|I_{n-k}] \tag{3}$$

Thus when H is multiplied with a codeword $C(t)$ the result should be 0.

$$HC(t) = 0 \tag{4}$$

LDPC codes are characterized by the utilization of a parity check matrix that has a low density of ones and thus making it a sparse matrix. The rate r is determined by the fraction $\frac{k}{n}$ and defines the parity check matrix's dimensions. Regular LDPC codes have the same weight of ones in every column and row of the parity check matrix and irregular LDPC codes are the ones that they do not have this property.

The Consultative Committee for Space Data Systems (CCSDS) standardizes the usage of irregular LDPC codes, which have demonstrated improved performance and can approach the Shannon Capacity limit. These codes are extensively employed in satellite communications, where higher-rate codes are preferred to limit bandwidth expansion for near-earth applications. In contrast, lower-rate codes, even with a rate of $\frac{1}{2}$, are commonly used in deep space applications that require greater signal-to-noise ratio (SNR) performance. However, the decoding of lower-rate codes demands intense computational resources due to the extended distance between the transmitter and receiver.

The process of generating a parity check matrix according to CCSDS will be demonstrated below.

1.2 LDPC Parity Check Matrix according to CCSDS

The LDPC code was chosen to have an information block length of 4096 bytes and a rate of 4/5, which is the most complex parity check matrix defined in [4]. The document mentions that for these parameters, rectangular MxM submatrices will be used. For $k = 4096, r = \frac{4}{5}$, the size of the submatrices is defined to me $M = 512$

The parity check matrix will be constructed with the combination of 3 types of submatrices. I_M is the $M \times M$ identity matrix, 0_M is the $M \times M$ zero matrix and Π_K is a permutation matrix. The permutation matrices have a 1 value for every row and column. The ones are distributed with the following formula:

For every i row that $i\epsilon\{0, M-1\}$ we calculate the column of the 1 value:

$$\pi_k(i) = \frac{M}{4}((\theta_k + \lfloor 4i/M \rfloor) \mod 4 + (\phi_k(\lfloor 4i/M \rfloor, M) + i) \mod M/4 \tag{5}$$

The variables θ_k and ϕ_k are provided in tables in the document. These values are used to iterate the index of the columns that the ones are positioned. The result is a permutation matrix where the 1s start from a column and the 1s in the following rows are located in the next column until a 1 has already been placed previously in the same column.

As the dimensions of these matrices are too large to allow a 1:1 presentation here, a smaller example is presented.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then the parity check matrix is constructed as a combination of the previously mentioned matrices according to:

$$\begin{bmatrix} 0_M & 0_M & 0_M & 0_M & 0_M & 0_M & 0_M & 0_M & I_M & 0_M & I_M \oplus \Pi_1 \\ \Pi_{21} \oplus \Pi_{22} \oplus \Pi_{23} & I_M & \Pi_{15} \oplus \Pi_{16} \oplus \Pi_{17} & I_M & \Pi_9 \oplus \Pi_{10} \oplus \Pi_1 1 & I_M & I_M & I_M & 0_M & I_M & \Pi_2 \oplus \Pi_3 \oplus \Pi_4 \\ I_M & \Pi_{24} \oplus \Pi_{25} \oplus \Pi_{26} & I_M & \Pi_{18} \oplus \Pi_{19} \oplus \Pi_{20} & I_M & \Pi_{12} \oplus \Pi_{13} \oplus \Pi_{14} & I_M & \Pi_5 \oplus \Pi_6 & 0_M & \Pi_7 \oplus \Pi_8 & I_M \end{bmatrix}$$

From this representation it is clear that the sum of ones in every row and columns are different. For example in the first M rows the sum of every element in each row will be 3 but in the following M rows it will be 18.

The parity check matrix, using the above formula and the corresponding generator matrix, using 2, are generated in the MATLAB script *calculateH45.m*. *calculateP.m* is used internally to calculate the permutation matrices.

2 Decoding Linear Codes using the Sum-Product Algorithm

2.1 Introduction

The Sum-Product algorithm [2],[5] is a widely used message-passing algorithm for decoding linear codes, also used extensively in LDPC decoders. The algorithm iteratively computes and updates the probability distributions of the bits in the codeword, based on the noisy received signal and the structure of the parity check matrix. The algorithm is based on the idea of representing the LDPC code as a Tanner graph, which is a bipartite graph that shows the connections between the variable nodes (representing the bits in the codeword) and the check nodes (representing the parity checks of the code).

The sum-product algorithm proceeds by passing messages along the edges of the Tanner graph, with each message representing a probability distribution over the values of the corresponding variable node or check node. The messages are updated using the sum-product rule, which involves taking the product of the incoming messages and then summing over the other variable or check nodes connected to the node in question. This process is repeated until a convergence criterion is met or a maximum number of iterations is reached. At the end of the decoding process, the algorithm outputs an estimate of the most likely codeword that could have generated the received signal.

The sum-product algorithm is known to be an efficient and effective method for LDPC decoding, and has been widely studied and applied in various fields, including digital communications, information theory, and coding theory. Its effectiveness is due in part to the fact that it can take advantage of the inherent sparsity of the LDPC code structure, and can often achieve near-optimal decoding performance with relatively low computational complexity.

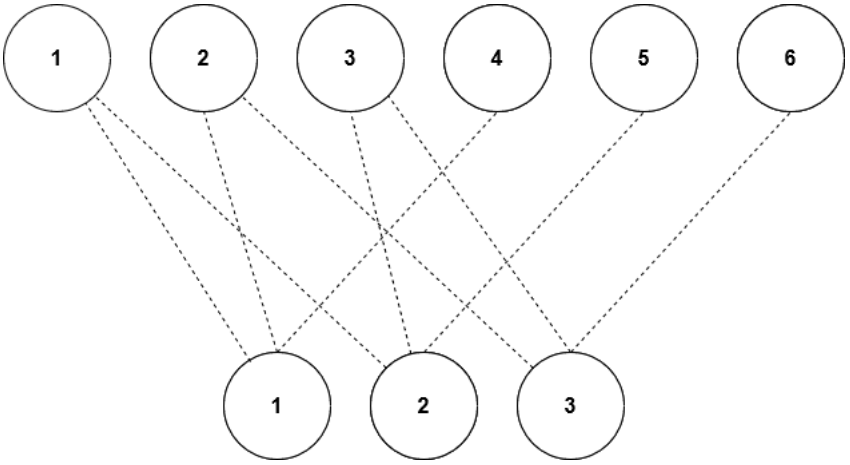

Figure 2: Tanner Graph for the Parity Check matrix from Section 2.2

Algorithm 1 Sum Product Algorithm

init L, $L_i = L(y_i|x_i)$
while $i \leq max - iter$ **do**
  $M \leftarrow L * H$
  $E_{i,j} \leftarrow ln\frac{1+\prod_{k=1,k\neq j}^{n} tanh(\frac{M_{i,k}}{2})}{1-\prod_{k=1,k\neq j}^{n} tanh(\frac{M_{i,k}}{2})}$
  if $L_i < 0$ **then**
    $c_i \leftarrow 1$
  else if $L_i > 0$ **then**
    $c_i \leftarrow 0$
  end if
  if $HC^T = 0$ **then**
    stop
  else if $HC^T \neq 0$ **then**
    continue
  end if
end while

2.2 Algorithm

We use a simple parity check matrix for clarification.

$$H = [P^T|I_{n-k}] = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

For illustration, we assume the original code word

$$X = (1, 1, 0, 0, 1, 1)$$

This was generated by encoding the message $(1, 1, 0)$ using the corresponding Generator Matrix

$$G = [I_k|P] = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

and the codeword we receive has one altered bit due to the channel:

$$Y = (1, 1, 0, 1, 1, 1)$$

. And we assume that the message is transformed with a binary symmetric channel (BSC) with the flip probability $p_e = 0.2$.

In a BSC, the received symbol y_i is a binary value that is flipped with probability p_e. If the transmitted bit is $y_i = 0$, then x_i is flipped with probability p_e and remains 0 with probability $1 - p_e$. Similarly, if $y_i = 1$, then x_i is flipped with probability $1 - p_e$ and becomes 1 with probability p_e. Therefore, we have:

$$P(x_i = 1|y_i = 0) = p_e$$
$$P(x_i = 1|y_i = 1) = 1 - p_e$$

These probabilities can be used to calculate the log likelihood ratio (LLR):

$$\begin{aligned} LLR_i(0) &= log(\frac{P(x_i = 0|y_i = 0)}{P(x_i = 1|y_i = 0)}) \\ &= log(\frac{1 - p_e}{p_e}) \\ &= 1.386 \\ LLR_i(1) &= log(\frac{P(x_i = 0|y_i = 1)}{P(x_i = 1|y_i = 1)}) \\ &= log(\frac{p_e}{1 - p_e}) \\ &= -1.386 \end{aligned}$$

If the $LLR < 0$ we can say that for current bit y, the original bit x should be 1. Then we can initialize thee original total LLR vector:

$$L_{total} = [-1.386, -1.386, 1.386, -1.386, -1.386, -1.386]$$

The initialized total LLR vector quantified under the binary symmetric channel for every received bit, the probability for the original code bit to be 0 or 1.

From the parity check matrix we know that the correct code must satisfy the following equations:

$$x_1 \oplus x_2 \oplus x_4 = 0$$
$$x_1 \oplus x_3 \oplus x_5 = 0$$
$$x_2 \oplus x_3 \oplus x_6 = 0$$

We use the first equation for the following illustration. Because each code information is independent, we can have the LLR:

$$\begin{aligned} LLR(x_1 \oplus x_2 \oplus x_4 = 0) &= log\frac{P(x_1 \oplus x_2 \oplus x_4 = 0, x_1 = 0|Y)}{P(x_1 \oplus x_2 \oplus x_4 = 0, x_1 = 1|Y)} \\ &= log(\frac{P(x_1 = 0|Y)P(x_2 \oplus x_4 = 0|Y)}{P(x_1 = 1|Y)P(x_2 \oplus x_4 = 1|Y)} \\ &= log(\frac{P(x_1 = 0|Y)}{P(x_1 = 1|Y)} + log(\frac{P(x_2 \oplus x_4 = 0|Y)}{P(x_2 \oplus x_4 = 1|Y)} \end{aligned}$$

We already have the initialized the LLR part:

$$LLR_{init}(x_1 \oplus x_2 \oplus x_4 = 0) = log(\frac{P(x_1 = 0|Y)}{P(x_1 = 1|Y)}$$

Therefore we want to calculate the LLR part:

$$LLR_{update}(x_1 \oplus x_2 \oplus x_4 = 0) = log(\frac{P(x_2 \oplus x_4 = 0|Y)}{P(x_2 \oplus x_4 = 1|Y)}$$

For an n-bit sequence, if each bit has the independent probability p_i to be 1, then the probability of that sequence to have an even number of 1s is

$$P(even) = \frac{1}{2}[1 + \prod_{i=1}^{n}(1 - 2p_i)]$$

So the problem for $P(x_1 \oplus x_2 \oplus ... \oplus x_n = 0, x_i = 0|Y)$, is equivalent to finding the probability that $x_i = 0$ and, simultaneously, the rest of sequence has even number of 1. We can transform the above equation to that

$$\begin{aligned} L(x_i|y_i) &= \frac{P(x_i = 0|y_i)}{P(x_i = 1|y_i)} \\ P(x_i = 0|y_i) &= \frac{L(x_i|y_i)}{1 + L(x_i|y_i)} \\ 2P(x_i = 0|y_i) - 1 &= \frac{L-1}{L+1} \\ &= tanh(\frac{1}{2} lnL(x_i|y_i)) \\ P(x_1 \oplus x_2 \oplus ...x_n = 0|Y) &= ln\frac{1 + \prod_{i=1}^{n} tanh(\frac{lnL(x_i|y_i)}{2})}{1 - \prod_{i=1}^{n} tanh(\frac{lnL(x_i|y_i)}{2})} \end{aligned}$$

Therefore, for the above example we can first have a matrix M that

$$M = \begin{bmatrix} -1.386 & -1.386 & 0 & -1.386 & 0 & 0 \\ -1.386 & 0 & 1.386 & 0 & -1.386 & 0 \\ 0 & -1.386 & 1.386 & 0 & 0 & -1.386 \end{bmatrix}$$

where $M_{i,j} = lnL(x_i|y_i)$ that is initialized with the binary symmetric channel. Then we can have the matrix E for calculating the LLR_{update} that is

$$E_{i,j} = ln\frac{1 + \prod_{k=1,k\neq j}^{n} tanh(\frac{M_{i,k}}{2})}{1 - \prod_{k=1,k\neq j}^{n} tanh(\frac{M_{i,k}}{2})}$$

For example, to satisfy the first equation:

$$x_1 \oplus x_2 \oplus x_4 = 0$$

$E_{i,j}$ stands for the LLR result for i_{th} equation and j_{th} bit of received y. For example, the $E_{1,1}$ it means that for the first bit of y the LLR function that satisfy the first equation $x_1 \oplus x_2 \oplus x_4 = 0$. we can get that

$$\begin{aligned} LLR_{update}(1,1) &= LLR_{update}(x_1 \oplus x_2 \oplus x_4 = 0) \\ &= log(\frac{P(x_2 \oplus x_4 = 0|y_1 = 1)}{P(x_2 \oplus x_4 = 1|y_1 = 1)} \\ &= ln\frac{1 + tanh(\frac{M_{1,2}}{2})tanh(\frac{M_{1,4}}{2})}{1 - tanh(\frac{M_{1,2}}{2})tanh(\frac{M_{1,4}}{2})} \\ &= 0.7538 \end{aligned}$$

We have the following matrix E

$$E = \begin{bmatrix} 0.7538 & 0.7538 & 0 & 0.7538 & 0 & 0 \\ -0.7538 & 0 & 0.7538 & 0 & -0.7538 & 0 \\ 0 & -0.7538 & 0.7538 & 0 & 0 & -0.7538 \end{bmatrix}$$

Each row of matrix E stands for the LLR function for one equation. To calculate the total LLR function, we can add up the individual elements of each column. Then we can calculate the total LLR_{update} for all the equations that need to be satisfied

$$LLR_{total} = L_{init} + L_{update}$$

We can have the following LLR_{total} vector

$$[-1.386, -1.386, 2.894, -0.632, -2.140, -2.140]$$

if the LLR is below zero we choose 1 as the result, and vice versa. We can see that the flipped bit likelihood is reducing with the updated information from the equation. We can have the decoded vector

$$[1, 1, 0, 1, 1, 1]$$

if $Hx^T = 0$ then we can stop the iteration, otherwise we use the above LLR information as the initial LLR function and continue the iteration. After two iteration, we can get the correct result

$$[1, 1, 0, 0, 1, 1]$$

A simple implementation of the Sum-Product algorithm in python can be found inside the *decoding.py* file. The function is used to illustrate the above numerical example in code as well.

The same function is used to decode a message encoded using the CCSDS defined Generator Matrix from Section 1.2 (stored in *data.mat* in *decodingCCSDS.py*. Two bitflips were applied to the codeword and are corrected using the Sum-Product algorithm.

References

[1] R.G. Gallager, ”Low-Density Parity-Check Codes”

[2] D.J.C. MacKay, ”Information Theory, Inference, and Learning Algo- rithm

[3] CCSDS GREEN BOOK 130.1-G-3 "TM SYNCHRONIZATION AND CHANNEL CODING—SUMMARY OF CONCEPTAND RATIONALE"

[4] CCSDS BLUE BOOK CCSDS 131.0-B-4 "TM SYNCHRONIZATION AND CHANNEL CODING"

[5] Alex Balatsoukas-Stimming, Telecommunications Laboratory, "Decoding of LDPC codes using the Sum-Product algorithm for the AWGN channel with BPSK modulation", Presentation Slides