Sum-Product Decoding

Abstract

Sum-product algorithm and max-product algorithm are generally used in the area of decoding. In this project, we build the AWGN channel and implement two decoding algorithm utilizing factor graph theory. Simulations are given for further analysis and conclusions.

1. Introduction

Sum-product algorithm, which is also known as belief propagation, is a message passing algorithm for performing inference on specific graphical models. The probability function is written as:

$$p(\mathbf{x}) = \prod_{a \in F} f_a(\mathbf{x}_a)$$

2. Background

Hamming(7,4) is a linear error-correcting code that encodes four bits of data into seven bits by adding three parity bits. The goal of Hamming is to generate a set of parity bits that overlap such that a single-bit error in a data bit or a parity bit can be detected and corrected. Hamming codeword consists of 7 bits(x1,x2,…,x7) satisfying equations as follow:

$$x4 + x5 + x6 + x7 = 0 \bmod 2;$$
$$x2 + x3 + x6 + x7 = 0 \bmod 2;$$
$$x1 + x3 + x5 + x7 = 0 \bmod 2;$$

Therefore, the parity-check matrix H can be defined as:

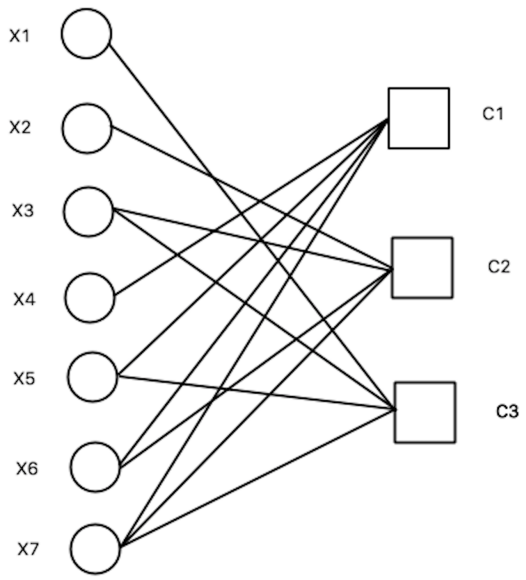
$$\mathbf{H} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Addictive white Gaussian noise(AWGN) is a basic noise model used in information theory to mimic the effect of many random processes. In a communication system, AWGN is often used as a channel model in which the only impairment to communication is a linear addition of wideband or white noise with a constant spectral density. Gaussian noise has a probability density function equal to that of normal distribution, which is defined as follow:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

3. Method

In this project, we mainly focus on building a decoder for Hamming(7,4) code. To implement sum-product algorithm and max-product algorithm, a factor graph corresponding to the parity-check matrix is vital for analysis on further calculations. According to Hamming parity-check matrix H, there are three nodes for information message and seven nodes for codewords. Hence, we can have the following factor graph:



We assume the parity for the communication system is even parity. Thus, to satisfy the criterion of Hamming(7,4) code, the probability calculation depends on all codeword nodes communicated to the same check node. For example, there are four nodes(x4,x5,x6,x7) connected to C1. The probability for x4 to be 1 or 0 using sum-product algorithm is:

P(x4=1) = P(x5=1) P(x6=1) P(x7=1) + P(x5=1) P(x6=0) P(x7=0) +
P(x5=0) P(x6=1) P(x7=0) + P(x5=0) P(x6=0) P(x7=1)

P(x4=0) = P(x5=0) P(x6=0) P(x7=0) + P(x5=1) P(x6=1) P(x7=0) +
P(x5=0) P(x6=1) P(x7=1) + P(x5=1) P(x6=0) P(x7=1)

With max-product algorithm, the probability for x4 should be:

P(x4=1) = Max[P(x5=1) P(x6=1) P(x7=1) , P(x5=1) P(x6=0) P(x7=0),
P(x5=0) P(x6=1) P(x7=0) , P(x5=0) P(x6=0) P(x7=1)]

P(x4=0) = Max[P(x5=0) P(x6=0) P(x7=0) , P(x5=1) P(x6=1) P(x7=0),
P(x5=0) P(x6=1) P(x7=1) , P(x5=1) P(x6=0) P(x7=1)]

For the error analysis, we use the following index:

$$P_e = \frac{\sum_{i=1}^{M} m_i}{7M}$$

4. Implementation

The general procedure of building this decoder consists of six phases. First of all, the decoder should be initialized in the runtime environment. Secondly, the AWGN channel is activated and generates random Gaussian noise to simulate the received codewords before initializing probabilities for all result sets. Thirdly, MPAdecoder simulates the decoding calculation using max-product algorithm and generate the most likely probabilities values for each codeword nodes. Fourthly, SPAdecoder simulates the decoding calculation using sum-product algorithm and generate the most likely probabilities values for

each codeword nodes. Fifthly, error code is counted and thus bit error probability is analyzed and demonstrated. At the end of the procedure, all used memories for the program are released and all data sets are cleared for the next loop.

4.1 Initializer

The function Initializer() sets up values for data sets. It creates a mapping between codeword[] and transCode[], which means 1 in transCode[] indicates 0 in codeword[] and -1 in transCode[] indicates 1 in codeword[]. Furthermore, it creates the factor graph for Hamming(7,4) code by restoring indexes of all codeword nodes communicated with three check nodes. In addition, different codeword nodes may communicate with more than one check node and thus have more than one probability calculation process in the decoding phase.

4.2 GaussianNoiseGenerator

GaussianNoiseGenerator() mainly creates random initial values for probability calculation according to:

$$recvCode = transCode + GaussianNoise$$

It simulates the noise results with normal distribution for MPAdeocoder and SPAdecoder. The calculated values are then restored in recvCode[]. In particular, the result sets for SPA is recalculated with the logarithm operation in order to avoid overflow and underflow.

4.3 MPAdecoder

MPAdecoder is the implemented decoder with max-product algorithm. With a finite runtime and samples, the probabilities for

codewords are calculated in MPA() functions based on the factor graph structure generated in the initializing phase. Codeword node x1, x2, x4 pass messages to only one check node while codeword node x3, x5, x6 communicate with two check nodes and there are three check nodes communicated with codeword node x7. In function MPA(), the main idea is to repeatedly choose the biggest probability among all the possible codeword solutions. Normalizing the calculated results is essential before comparing probabilities with the stored old values. It uses MAX() to decide the bigger value during comparison.

4.4 SPAdecoder

SPAdecoder is the implemented decoder with sum-product algorithm. With a finite runtime and samples, the probabilities for codewords are calculated in SPA() functions based on the factor graph structure generated in the initializing phase. It functions in a similar way as MPAdecoder in terms of dealing with codeword nodes. In function SPA(), it sums up all the result values according to all solutions among all other communicated codeword nodes. Since the summation values go higher up during the calculation loop, the function LOGSUM() is therefore to lower the values. Otherwise the program will crash because of the overflow situation.

4.5 ErrorAnalyzer

To evaluate the performance of two implemented decoders, the errorAnalyzer is important to collect the error information including the total number of all falsely predicted codewords and bit error probability. The main idea is to compare the calculated probabilities for code 0 and code 1 in the MPAresults and SPAresults. If the origin code is 0 and the probability for this code to be 0 is lower than that of being 1, it generates an error count, and vice versa.

4.6 Destructor

At the end of the procedure, error counts and result sets should be cleared to avoid the reuse of calculated data which is likely to generate the same result in the following loop.

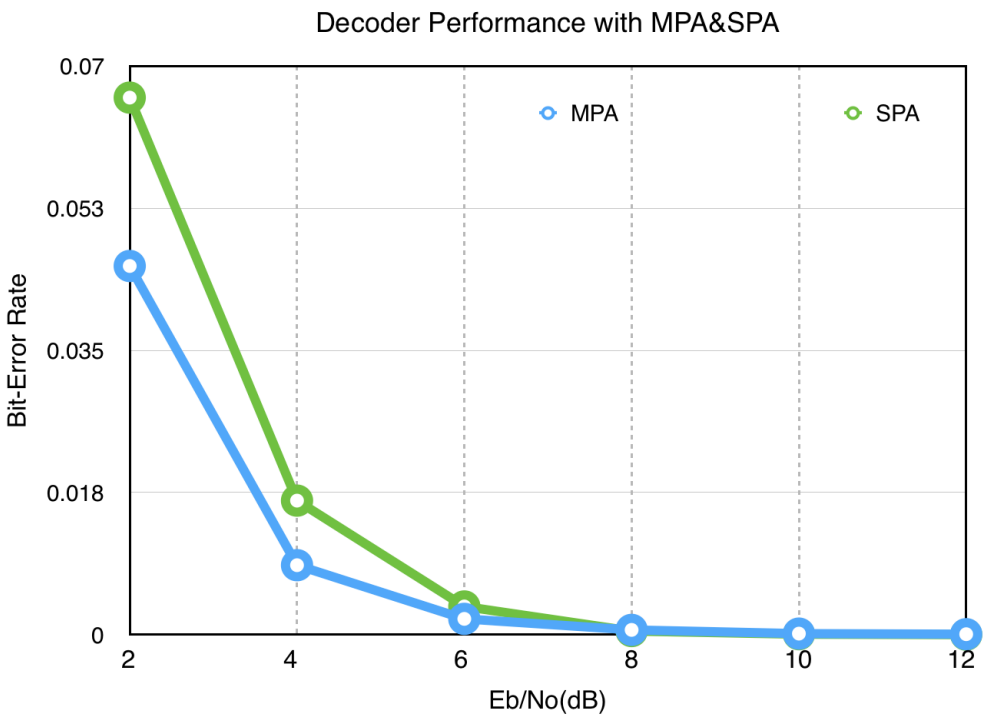
5. Results & Performance

For performance evaluation, we use 10000 samples of codewords in the simulation with a runtime loop of 1000 times. In addition, we choose different channel noise variances(1, 0.5, 0.25, 0.125) for the test and set the origin codeword as 1010101. At the same time, the mean value for the distribution function is 0. The test result demonstrates as follow:

```
Codeword: 1010101

Variance: 1
Number of bit error for Max Product Algorithm: 9467
Number of bit error for Sum Product Algorithm: 10849
Bit error probability for Max Product Algorithm: 13.5243%
Bit error probability for Sum Product Algorithm: 15.4986%

Variance: 0.5
Number of bit error for Max Product Algorithm: 3925
Number of bit error for Sum Product Algorithm: 5352
Bit error probability for Max Product Algorithm: 5.60714%
Bit error probability for Sum Product Algorithm: 7.64571%

Variance: 0.25
Number of bit error for Max Product Algorithm: 896
Number of bit error for Sum Product Algorithm: 1575
Bit error probability for Max Product Algorithm: 1.28%
Bit error probability for Sum Product Algorithm: 2.25%

Variance: 0.125
Number of bit error for Max Product Algorithm: 75
Number of bit error for Sum Product Algorithm: 90
Bit error probability for Max Product Algorithm: 0.107143%
Bit error probability for Sum Product Algorithm: 0.128571%

Program ended with exit code: 0
```

To further analyze the bit error rate, we set Eb/No as 2, 4, 6, 8, 10, 12 dB and hence we have the following simulation results and corresponding plots

```
Codeword: 1010101

Variance: 0.4375
Number of bit error for Max Product Algorithm: 3177
Number of bit error for Sum Product Algorithm: 4628
Bit error probability for Max Product Algorithm: 4.53857%
Bit error probability for Sum Product Algorithm: 6.61143%

Variance: 0.21875
Number of bit error for Max Product Algorithm: 599
Number of bit error for Sum Product Algorithm: 1155
Bit error probability for Max Product Algorithm: 0.855714%
Bit error probability for Sum Product Algorithm: 1.65%

Variance: 0.145833
Number of bit error for Max Product Algorithm: 136
Number of bit error for Sum Product Algorithm: 243
Bit error probability for Max Product Algorithm: 0.194286%
Bit error probability for Sum Product Algorithm: 0.347143%

Variance: 0.109375
Number of bit error for Max Product Algorithm: 41
Number of bit error for Sum Product Algorithm: 31
Bit error probability for Max Product Algorithm: 0.0585714%
Bit error probability for Sum Product Algorithm: 0.0442857%

Variance: 0.0875
Number of bit error for Max Product Algorithm: 9
Number of bit error for Sum Product Algorithm: 4
Bit error probability for Max Product Algorithm: 0.0128571%
Bit error probability for Sum Product Algorithm: 0.00571429%

Variance: 0.0729167
Number of bit error for Max Product Algorithm: 4
Number of bit error for Sum Product Algorithm: 0
Bit error probability for Max Product Algorithm: 0.00571429%
Bit error probability for Sum Product Algorithm: 0%

Program ended with exit code: 0
```



6. Conclusion

Sum-product algorithm and max-product algorithm provide theoretical influences on the construction of general decoder in a more precise way. From our simulations, the bit error probabilities for these two algorithms are relatively low. As the value of noise variance decreases, the bit error probability drops dramatically down to around 0.1%, which means the decoding process is nearly faultless. Additionally, MPAdecoder has a sightly better performance than SPAdecoder in most cases during the simulations.

7. Reference

[1] F.R. Kschishang and B.J. Frey, “Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models,” IEEE JSAC, vol.16, no.2, Feb. 1998.

[2]F.R.Kschishang,B.J.Frey,andH.Loeliger,“FactorGraphsandtheSum- Product Algorithm,” IEEE Trans. on Information Theory, vol. 47, no. 2, Feb. 2001.

[3]Hari Sankar,Krishna R. Narayanan,“Memory-Efficient Sum–Product Decoding of LDPC Codes”,IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 52, NO. 8, AUGUST 2004