Hand-in 3

Zhongwang Fu, Personnumber: 200203237532, 2025.3.2

Part 1: A simple example code

The bit-flipping decoding algorithm follows an iterative process to correct errors in the received vector based on the parity-check matrix. The key steps involved in the decoding process are as follows:

1. Input: received vector r, Initialize: $\hat{v} = r$

2. Syndrome Computation: $s^{(i)} = \hat{v} \cdot H^T$ stop if $s^{(i)} = 0$

3. Compute number of invalid syndromes per codebit:

$\sigma^{(i)} = (\sigma_1, \sigma_2, \dots, \sigma_n)$, where $\sigma_c = \sum_{r=1}^{n-k} h_{r,c} \cdot s_r^i$, $h_{r,c}$: colum c of H

4. Bit Flipping: The bits with the maximum error contribution are flipped.

5. Iteration: Steps 2-4 are repeated until the syndrome becomes zero or the maximum iteration limit is reached.

6. Output: The final decoded codeword $\hat{v}$ is returned.

We consider first the code defined by the parity-check matrix used in Lecture 9:

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

Implement the bit-flipping decoder and decode the sequence

$$r = (1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1)$$

The bit-flipping decoding process proceeds as follows:

1.First Iteration:

Initial codeword: $\hat{v}^{(0)} = r = (1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1)$

Error contribution vector: $\sigma^{(1)} = (1 \quad 3 \quad 2 \quad 1 \quad 3 \quad 1 \quad 1)$

Flipping bits at positions 2 and 5, codeword estimate: $\hat{v}^{(1)} = (1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1)$

2.Second Iteration:

Error contribution vector: $\sigma^{(2)} = (0 \quad 1 \quad 1 \quad 2 \quad 2 \quad 0 \quad 0)$

Flipping bits at positions 4 and 5, codeword estimate: $\hat{v}^{(2)} = (1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1)$

3.Third Iteration:

Error contribution vector: $\sigma^{(3)} = (0 \quad 1 \quad 1 \quad 2 \quad 4 \quad 1 \quad 1)$

Flipping bits at position 5, codeword estimate: $\hat{v}^{(3)} = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$

Syndrome becomes zero, decoding terminates.

The final decoded codeword is $\boldsymbol{\hat{v}} = (\mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1})$, **3 iterations** are required for decoding.

Part 2: Decoding a longer code

A longer text is transmitted using a systematic (1024, 512) LDPC code over a BSC. The received text after transmission is:

Received Text (parity bits are shown here): **“Lmfe ic"what hQppejc0when y'5 a2á$busy mIkinc oôhes plans.**

fÆ□YÎË¥êH□Þ¹□«ÕçäØ□°áÄ□©^°3jÓpð}«Ð³gNrG<YÒäíÆÏ =7>Û□”

The LDPC code is defined by the parity-check matrix provided in the file H_1024_3_6.txt. The corresponding binary received sequence is loaded from ChannelOutputBinary.txt and decoded using the bit-flipping decoder. After decoding, the binary sequence is converted back into text, assuming 8 bits per character.

The bit-flipping decoding process converged after **13 iterations**. The final decoded text is:

Decoded Text: “**Life is what happens when you are busy making other plans. AÆ□yÎÁ¥êÈ□Ê¹□O□ÕçæÙ□°áÄ□¡^°3zÓpðy□Ð±gNrG<YÒäí æÏ=7>Û□**”

Part 3: Simulation of bit error performance

The plotted results shown in figure 1 illustrate the bit error rate (BER) performance of the (1024,512) LDPC code under bit-flipping decoding.

The uncoded transmission BER (yellow dashed line), which follows $P_b = \epsilon$, representing the theoretical bit error probability without coding. The block error probability P_B (blue circles) and bit error probability P_b (red crosses) for the LDPC-coded system.

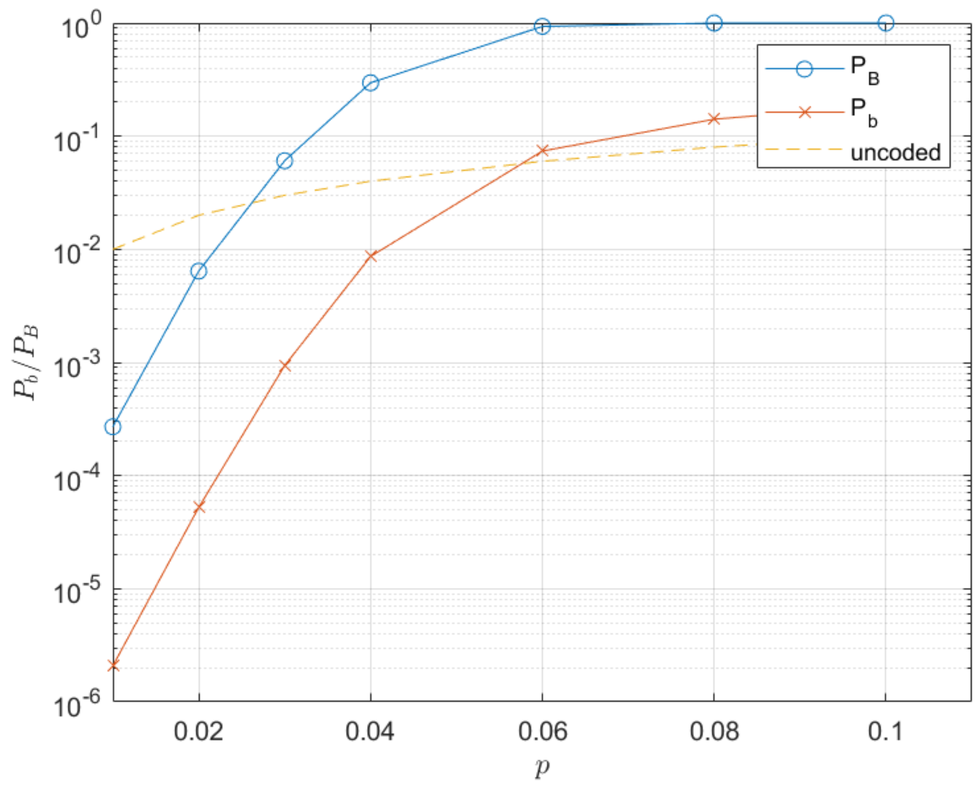

Fig 1. Bit error rate and block error rate of BFD

We can see that below $\epsilon = 0.06$, errors are sparse enough for the decoder to correct them successfully. However, beyond this point, the number of errors becomes too large for iterative correction to work effectively. The performance drop is particularly evident in the block error probability (P_B), which approaches 1 at $\epsilon = 0.06$, indicating that nearly every transmitted block contains errors after decoding. A $\epsilon = 0.08$, $P_B = 1$, meaning the decoder fails entirely—none of the received codewords are correctly decoded.

This performance may because of the decoder is limited to a maximum of 25 iterations. As ϵ increases, more iterations are required for successful decoding. When the error rate is too high, even after 25 iterations, the decoder still cannot correct all errors, leading to an early stopping condition with a high residual BER.

Appendix

1. Part 1 & 2

```
clc;
clear;

%% Part1
H = [1, 1, 0, 0, 0, 0, 0; ...
      0, 1, 1, 0, 0, 0, 0; ...
      0, 1, 1, 1, 1, 0, 0; ...
      0, 0, 0, 1, 1, 0, 0; ...
      0, 0, 0, 0, 1, 1, 0; ...
      0, 0, 0, 0, 1, 0, 1];

r = [1, 0, 1, 0, 0, 1, 1];

i_max = 25;

bit_flipping_decoder(H, r, i_max);

%% Part2
H = readmatrix('H_1024_3_6.txt');
r = readmatrix('ChannelOutputBinary.txt');

num_chars = length(r) / 8;
decoded_text = char(bin2dec(num2str(reshape(r(1:num_chars*8), 8, [])')))');
fprintf('Received Text:\n%s\n', decoded_text);

[decoded_bits, num_iter] = bit_flipping_decoder(H, r, i_max);
```

```
    num_chars = length(decoded_bits) / 8;
    decoded_text = char(bin2dec(num2str(reshape(decoded_bits(1:num_chars*8), 8,
[])')))');
    fprintf('Decoded Text:\n%s\n', decoded_text);


    function [decoded_bits, num_iterations] = bit_flipping_decoder(H, r, i_max)
        hat_v = r;
        iter = 0;
        s = mod(hat_v*H', 2);

        [k, n] = size(H);
        sigma = zeros(1, n);

        while ~all(s == 0) && iter < i_max
            iter = iter + 1;

            for c = 1:n
                sigma(c) = sum(s'.*H(:, c));
            end

            flip_idx = find(sigma == max(sigma));
            hat_v(flip_idx) = mod(hat_v(flip_idx)+1, 2);
            s = mod(hat_v*H', 2);

            fprintf('\\hat{v}^{(%d)} = [%s]\n', iter, num2str(hat_v));
            fprintf('\\sigma^{(%d)} = [%s]\n', iter, num2str(sigma));

        end

        fprintf('decoded codeword: %s\n', num2str(hat_v));
        fprintf('Total iterations: %d\n', iter);

        decoded_bits = hat_v;
        num_iterations = iter;
end
```

2. Part 3

```
% Simulation example with repetition code
%
% EITN45 Information Theory
% Lund University
%
% created by Michael Lentmaier 2024-02-12
clear;
clc;

n = 1024;
k = 512;
R=k/n;
i_max = 25;
H = readmatrix('H_1024_3_6.txt');

v=zeros(1,n);

pvec=[0.01 0.02 0.03 0.04 0.06 0.08 0.1]; %[0.001 0.005 0.01 0.005 0.1 0.5];

% Reset error counters
numbits=zeros(1,length(pvec));
biterrors=zeros(1,length(pvec));
blockerrors=zeros(1,length(pvec));

disp('Simulation running ...');
numBlocks=100000;

parfor indp=1:length(pvec)

    for block=1:numBlocks

        errors=(rand(1,n)<pvec(indp));
        r=mod(v+errors,2);

        %% repetition decoder (replace with your decoder)

        % uhat=(sum(r)>n/2);
        % vhat=repmat(uhat,1,n);
        [vhat, num_iter] = bit_flipping_decoder(H, r, i_max);

        %% end decoder

        errorpositions=(vhat~=v);
        biterrors(indp) =    biterrors(indp) + sum(errorpositions);
        blockerrors(indp) = blockerrors(indp) + any(vhat~=v);
        numbits(indp) = numbits(indp) + n;
    end
end
disp('Ready.');
Pb=biterrors./numbits;
Pblock=blockerrors/numBlocks;

figure
semilogy(pvec,Pblock,'-o'); hold on;
semilogy(pvec,Pb,'-x');
semilogy(pvec,pvec,'--'); % uncoded
xlabel('$p$','interpreter','latex');
ylabel('$P_b / P_B$','interpreter','latex');
grid on;
legend('P_B','P_b','uncoded');

%save 'SimExample.mat' pvec Pblock Pb

function [decoded_bits, num_iterations] = bit_flipping_decoder(H, r, i_max)
        hat_v = r;
        iter = 0;
        s = mod(hat_v*H', 2);
        [k, n] = size(H);
        sigma = zeros(1, n);

        while ~all(s == 0) && iter < i_max
            iter = iter + 1;

            for c = 1:n
                sigma(c) = sum(s'.*H(:, c));
            end

            flip_idx = find(sigma == max(sigma));
            hat_v(flip_idx) = mod(hat_v(flip_idx)+1, 2);
            s = mod(hat_v*H', 2);

            % fprintf('\\sigma^{(%d)} = [%s]\n', iter, num2str(sigma));
            % fprintf('\\hat{v}^{(%d)} = [%s]\n', iter, num2str(hat_v));

        end
        % fprintf('decoded codeword: %s\n', num2str(hat_v));
        % fprintf('Total iterations: %d\n', iter);

        decoded_bits = hat_v;
        num_iterations = iter;
end
```