ECE F344

Information Theory and Coding

AY 2024-25

Sem 2

Assignment 3

Viterbi Decoding

Submitted by

Lakshya Jain

2022AAPS0247P

Introduction to Trellis and Viterbi algorithms

A trellis diagram provides a visual representation of the state transitions in a finite state machine over time. In the context of convolutional codes, these diagrams illustrate how encoded outputs are generated from input sequences and how the encoder's internal states evolve. Each path through the trellis represents a possible encoded sequence, making it an essential tool for understanding and decoding convolutional codes.

The Viterbi algorithm, developed by Andrew Viterbi in 1967, is a dynamic programming approach that efficiently finds the most likely sequence of hidden states in a Markov model. When applied to convolutional codes, it determines the most probable transmitted sequence by evaluating all possible paths through the trellis and selecting the one with the minimum cumulative error metric. This makes it optimal for maximum likelihood decoding of convolutional codes in the presence of noise.

Implementation Overview

This project implements convolutional encoding and Viterbi decoding to demonstrate error correction capabilities. The system processes a binary sequence "1011000" through a rate 1/2 convolutional encoder characterized by generator polynomials g1=110 and g2=111. These generators define how each input bit affects the encoder's output, with each element representing a connection to the corresponding shift register tap.

The encoding process transforms the 7-bit input into a 14-bit output sequence by mapping each input bit to two output bits based on the current state of the encoder's memory registers. The trellis structure visualizes all possible state transitions that occur during encoding.

On the receiving end, when the bit sequence "01 11 01 11 01 01 11" arrives, the Viterbi decoder analyses potential paths through the trellis. For each time step and state, the algorithm calculates:

- Branch metrics: The Hamming distance between expected and received bit pairs
- Path metrics: The accumulated error along each potential path
- Survivor paths: The most likely preceding state for each current state

By tracing back through the survivor paths from the final state with minimum metric, the decoder reconstructs the original message sequence. The implementation provides detailed visibility into metric calculations and the decision-making process at each step, clearly illustrating how the algorithm converges to the optimal solution.

Code

```
clear all;
close all;
clc;

% message sequence to be encoded
message_sequence = [1 0 1 1 0 0 0];
disp('Original message sequence:');
disp(message_sequence);

constraint_length = 3;
code_generators = [6, 7];

% trellis structure
trellis_structure = poly2trellis(constraint_length, code_generators);

% Encoding
encoded_output = convenc(message_sequence, trellis_structure);
disp('Encoded bit sequence:');
disp(encoded_output);

%using encoded output as received sequence
received_sequence = encoded_output;
disp('Received sequence (no errors):');
disp(received_sequence);

% bit info
msg_length = length(message_sequence);
encoded_length = length(encoded_output);
fprintf('Message length: %d bits\n', msg_length);
fprintf('Encoded length: %d bits\n', encoded_length);

% built-in Viterbi decoder for verification
decoded_builtin = vitdec(received_sequence, trellis_structure, msg_length,
'trunc', 'hard');
```

```
disp('Decoded sequence (using built-in function):');
disp(decoded_builtin);

if isequal(message_sequence, decoded_builtin)
    disp('✓ VERIFICATION PASSED: Decoded sequence matches original message');
else
    disp('✗ VERIFICATION FAILED: Decoded sequence does not match original message');
    disp('Errors at positions:');
    disp(find(message_sequence ~= decoded_builtin));
end

%% Custom Viterbi Decoder Implementation

% decoder parameters
num_states = 2^(constraint_length-1);
time_steps = msg_length;
bits_per_symbol = 2;  % For rate 1/2 code

% received bits to symbols (pairs of bits)
rx_symbols = reshape(received_sequence, bits_per_symbol, [])';

% Viterbi algorithm data structures
path_metrics = Inf(time_steps+1, num_states);
path_metrics(1,1) = 0;  % Start at state 0 with metric 0
survivor_paths = zeros(time_steps, num_states);
branch_metrics = cell(time_steps, num_states, num_states);
valid_transitions = zeros(time_steps, num_states, num_states);

% Forward pass through trellis
disp('');
disp('FORWARD PASS - VITERBI ALGORITHM');
disp('===============================');

for t = 1:time_steps
    disp(['Time Step ' num2str(t) ':']);
```

```
    disp('Current    Input    Next     Output    Branch    Current    New
Selected');
    disp('State              State              Metric    Metric
Metric');
    disp('---------------------------------------------------------------------
------');

    for next_state = 0:num_states-1
        for current_state = 0:num_states-1
            for input_bit = 0:1
                % Check if this input leads to next_state
                if trellis_structure.nextStates(current_state+1,
input_bit+1) == next_state
                    % Mark valid transition
                    valid_transitions(t, current_state+1, next_state+1) =
1;

                    % Get expected output for this transition
                    output_code =
trellis_structure.outputs(current_state+1, input_bit+1);
                    expected_bits = dec2bin(output_code, bits_per_symbol) -
'0';

                    %  Hamming distance (branch metric)
                    branch_metric = sum(rx_symbols(t,:) ~= expected_bits);
                    branch_metrics{t, current_state+1, next_state+1} =
branch_metric;

                    %  new path metric
                    new_metric = path_metrics(t, current_state+1) +
branch_metric;

                    % Format output for display
                    output_str = [num2str(expected_bits(1))
num2str(expected_bits(2))];

                    % Display computation
                    if ~isinf(path_metrics(t, current_state+1))
                        fprintf('  %d        %d        %d       %s
%d        %.1f       %.1f', ...
```

```
                            current_state, input_bit, next_state,
output_str, branch_metric, ...
                            path_metrics(t, current_state+1), new_metric);

                        % Update path metric if better
                        if new_metric < path_metrics(t+1, next_state+1)
                            path_metrics(t+1, next_state+1) = new_metric;
                            survivor_paths(t, next_state+1) =
current_state;
                            fprintf('      ✓\n');
                        else
                            fprintf('\n');
                        end
                    end
                end
            end
        end
    end

    % path metrics after this time step
    disp(' ');
    disp(['Path Metrics after Time Step ' num2str(t) ':']);
    for s = 0:num_states-1
        if ~isinf(path_metrics(t+1, s+1))
            fprintf('  State %d: %.1f\n', s, path_metrics(t+1, s+1));
        else
            fprintf('  State %d: Inf\n', s);
        end
    end
    disp(' ');
end

% Traceback to find survivor path
disp('TRACEBACK - FINDING SURVIVOR PATH');
disp('================================');

%  state with minimum path metric at final time step
```

```
[~, best_state_idx] = min(path_metrics(time_steps+1, :));
final_state = best_state_idx - 1;

fprintf('Final state with minimum metric: State %d (Metric = %.1f)\n', ...
    final_state, path_metrics(time_steps+1, best_state_idx));

% Initializing traceback variables
state_sequence = zeros(1, time_steps+1);
state_sequence(end) = final_state;
decoded_output = zeros(1, time_steps);

% Performing traceback
disp(' ');
disp('Traceback Process:');
disp('Time    Current    Previous    Input');
disp('Step    State      State       Bit');
disp('--------------------------------');

for t = time_steps:-1:1
    current_state = state_sequence(t+1);
    previous_state = survivor_paths(t, current_state+1);
    state_sequence(t) = previous_state;

    % input bit that caused this transition
    for input_bit = 0:1
        if trellis_structure.nextStates(previous_state+1, input_bit+1) ==
current_state
            decoded_output(t) = input_bit;
            break;
        end
    end

    fprintf('  %d      %d           %d           %d\n', ...
        t, current_state, previous_state, decoded_output(t));
end
```

```
% Displaying custom decoder results
disp(' ');
disp('Custom decoder output:');
disp(decoded_output);

% Verifying custom decoder
if isequal(message_sequence, decoded_output)
    disp('✓ CUSTOM DECODER VERIFICATION PASSED');
else
    disp('✗ CUSTOM DECODER VERIFICATION FAILED');
    disp('Errors at positions:');
    disp(find(message_sequence ~= decoded_output));
end

%% Visualization of Trellis Diagram

% figure for trellis diagram
figure('Name', 'Trellis Diagram', 'Position', [100, 100, 900, 600]);
hold on;
grid off;
box on;

% Spacing parameters
time_spacing = 1;
state_spacing = 1.2;
node_size = 8;

% Plot all states at each time step
for t = 1:time_steps+1
    for s = 0:num_states-1
        % Plot node
        plot(t*time_spacing, -s*state_spacing, 'ko', 'MarkerFaceColor',
'k', 'MarkerSize', node_size);

        % Add path metric label
        if ~isinf(path_metrics(t, s+1))
```

```
            text(t*time_spacing, -s*state_spacing-0.25, sprintf('PM: %.1f',
path_metrics(t, s+1)), ...
                'FontSize', 8, 'HorizontalAlignment', 'center', 'Color',
'blue');
        end
    end
end

% Plot all transitions
for t = 1:time_steps
    for from_state = 0:num_states-1
        for to_state = 0:num_states-1
            if valid_transitions(t, from_state+1, to_state+1)
                % Find input bit for this transition
                for input_bit = 0:1
                    if trellis_structure.nextStates(from_state+1,
input_bit+1) == to_state
                        % Get output bits
                        output_code =
trellis_structure.outputs(from_state+1, input_bit+1);
                        output_bits = dec2bin(output_code,
bits_per_symbol);

                        % Get branch metric
                        bm = branch_metrics{t, from_state+1, to_state+1};

                        % Plot regular transition (gray dashed)
                        plot([t, t+1]*time_spacing, [-from_state, -
to_state]*state_spacing, ...
                            'k--', 'LineWidth', 0.8, 'Color', [0.7 0.7
0.7]);

                        % Add transition label
                        mid_x = (t + 0.5) * time_spacing;
                        mid_y = (-from_state - 0.5 * (to_state -
from_state)) * state_spacing;

                        label = sprintf('%d/%s (BM:%d)', input_bit,
output_bits, bm);
```

```
                        text(mid_x, mid_y+0.1, label, 'FontSize', 7,
'Color', 'black', ...
                            'HorizontalAlignment', 'center');

                        break;
                    end
                end
            end
        end
    end
end

% Highlight survivor path
for t = 1:time_steps
    current_state = state_sequence(t);
    next_state = state_sequence(t+1);

    % Plot survivor path with thick green line
    plot([t, t+1]*time_spacing, [-current_state, -
next_state]*state_spacing, ...
        'g-', 'LineWidth', 2.5);

    % Highlight nodes on the path
    plot(t*time_spacing, -current_state*state_spacing, 'go', ...
        'MarkerFaceColor', 'g', 'MarkerSize', node_size+2);
end
% Highlight final node
plot((time_steps+1)*time_spacing, -state_sequence(end)*state_spacing, 'go',
...
    'MarkerFaceColor', 'g', 'MarkerSize', node_size+2);

% Add annotations and formatting
title('Trellis Diagram with Path Metrics and Survivor Path');
xlabel('Time Step');
ylabel('State');
set(gca, 'YTick', -state_spacing * (0:num_states-1));
set(gca, 'YTickLabel', 0:num_states-1);
```

```
set(gca, 'XTick', 1:time_steps+1);
xlim([0.5, time_steps+1.5]);
ylim([-(num_states-0.5)*state_spacing, 0.5]);

%% Test with specific received sequence from assignment

% Given received sequence from assignment
specific_received = [0 1 1 1 0 1 1 1 0 1 0 1 1 1];
disp(' ');
disp('TESTING WITH SPECIFIC RECEIVED SEQUENCE');
disp('======================================');
disp('Assignment received bits:');
disp(specific_received);

% Validate length
expected_length = 2 * msg_length;
if length(specific_received) ~= expected_length
    fprintf('WARNING: Expected %d bits for %d input bits, but got %d
bits.\n', ...
        expected_length, msg_length, length(specific_received));
end

% Decode with built-in decoder
decoded_specific = vitdec(specific_received, trellis_structure, msg_length,
'trunc', 'hard');
disp('Decoded result from specific sequence:');
disp(decoded_specific);

% Verify against original message
if isequal(message_sequence, decoded_specific)
    disp('✓ Specific sequence decodes to original message');
else
    disp('✗ Specific sequence does not decode to original message');
    disp('Differences at positions:');
    disp(find(message_sequence ~= decoded_specific));
end
```

Output

```
Original message sequence:
     1     0     1     1     0     0     0

Encoded bit sequence:
     1     1     1     1     1     0     0     0     1     0     0     1     0     0

Received sequence (no errors):
     1     1     1     1     1     0     0     0     1     0     0     1     0     0

Message length: 7 bits
Encoded length: 14 bits
Decoded sequence (using built-in function):
     1     0     1     1     0     0     0

✓ VERIFICATION PASSED: Decoded sequence matches original message
```

```
FORWARD PASS - VITERBI ALGORITHM
================================
Time Step 1:
Current     Input    Next      Output    Branch     Current     New       Selected
State                State               Metric     Metric      Metric
------------------------------------------------------------------------
  0          0        0         00        2          0.0        2.0       ✓
  0          1        2         11        0          0.0        0.0       ✓

Path Metrics after Time Step 1:
  State 0: 2.0
  State 1: Inf
  State 2: 0.0
  State 3: Inf
```

```
Time Step 2:
Current     Input    Next      Output    Branch     Current     New       Selected
State                State               Metric     Metric      Metric
------------------------------------------------------------------------
  0          0        0         00        2          2.0        4.0       ✓
  2          0        1         11        0          0.0        0.0       ✓
  0          1        2         11        0          2.0        2.0       ✓
  2          1        3         00        2          0.0        2.0       ✓

Path Metrics after Time Step 2:
  State 0: 4.0
  State 1: 0.0
  State 2: 2.0
  State 3: 2.0
```

Time Step 3:

Current State	Input	Next State	Output	Branch Metric	Current Metric	New Metric	Selected
0	0	0	00	1	4.0	5.0	✓
1	0	0	01	2	0.0	2.0	✓
2	0	1	11	1	2.0	3.0	✓
3	0	1	10	0	2.0	2.0	✓
0	1	2	11	1	4.0	5.0	✓
1	1	2	10	0	0.0	0.0	✓
2	1	3	00	1	2.0	3.0	✓
3	1	3	01	2	2.0	4.0	

Path Metrics after Time Step 3:
State 0: 2.0
State 1: 2.0
State 2: 0.0
State 3: 3.0

Time Step 4:

Current State	Input	Next State	Output	Branch Metric	Current Metric	New Metric	Selected
0	0	0	00	0	2.0	2.0	✓
1	0	0	01	1	2.0	3.0	
2	0	1	11	2	0.0	2.0	✓
3	0	1	10	1	3.0	4.0	
0	1	2	11	2	2.0	4.0	✓
1	1	2	10	1	2.0	3.0	✓
2	1	3	00	0	0.0	0.0	✓
3	1	3	01	1	3.0	4.0	

Path Metrics after Time Step 4:
State 0: 2.0
State 1: 2.0
State 2: 3.0
State 3: 0.0

Time Step 5:

Current State	Input	Next State	Output	Branch Metric	Current Metric	New Metric	Selected
0	0	0	00	1	2.0	3.0	✓
1	0	0	01	2	2.0	4.0	
2	0	1	11	1	3.0	4.0	✓
3	0	1	10	0	0.0	0.0	✓
0	1	2	11	1	2.0	3.0	✓
1	1	2	10	0	2.0	2.0	✓
2	1	3	00	1	3.0	4.0	✓
3	1	3	01	2	0.0	2.0	✓

Path Metrics after Time Step 5:
State 0: 3.0
State 1: 0.0
State 2: 2.0
State 3: 2.0

Time Step 6:

Current State	Input	Next State	Output	Branch Metric	Current Metric	New Metric	Selected
0	0	0	00	1	3.0	4.0	✓
1	0	0	01	0	0.0	0.0	✓
2	0	1	11	1	2.0	3.0	✓
3	0	1	10	2	2.0	4.0	
0	1	2	11	1	3.0	4.0	✓
1	1	2	10	2	0.0	2.0	✓
2	1	3	00	1	2.0	3.0	✓
3	1	3	01	0	2.0	2.0	✓

Path Metrics after Time Step 6:
State 0: 0.0
State 1: 3.0
State 2: 2.0
State 3: 2.0

Time Step 7:

Current State	Input	Next State	Output	Branch Metric	Current Metric	New Metric	Selected
0	0	0	00	0	0.0	0.0	✓
1	0	0	01	1	3.0	4.0	
2	0	1	11	2	2.0	4.0	✓
3	0	1	10	1	2.0	3.0	✓
0	1	2	11	2	0.0	2.0	✓
1	1	2	10	1	3.0	4.0	
2	1	3	00	0	2.0	2.0	✓
3	1	3	01	1	2.0	3.0	

Path Metrics after Time Step 7:
State 0: 0.0
State 1: 3.0
State 2: 2.0
State 3: 2.0

TRACEBACK - FINDING SURVIVOR PATH
================================
Final state with minimum metric: State 0 (Metric = 0.0)

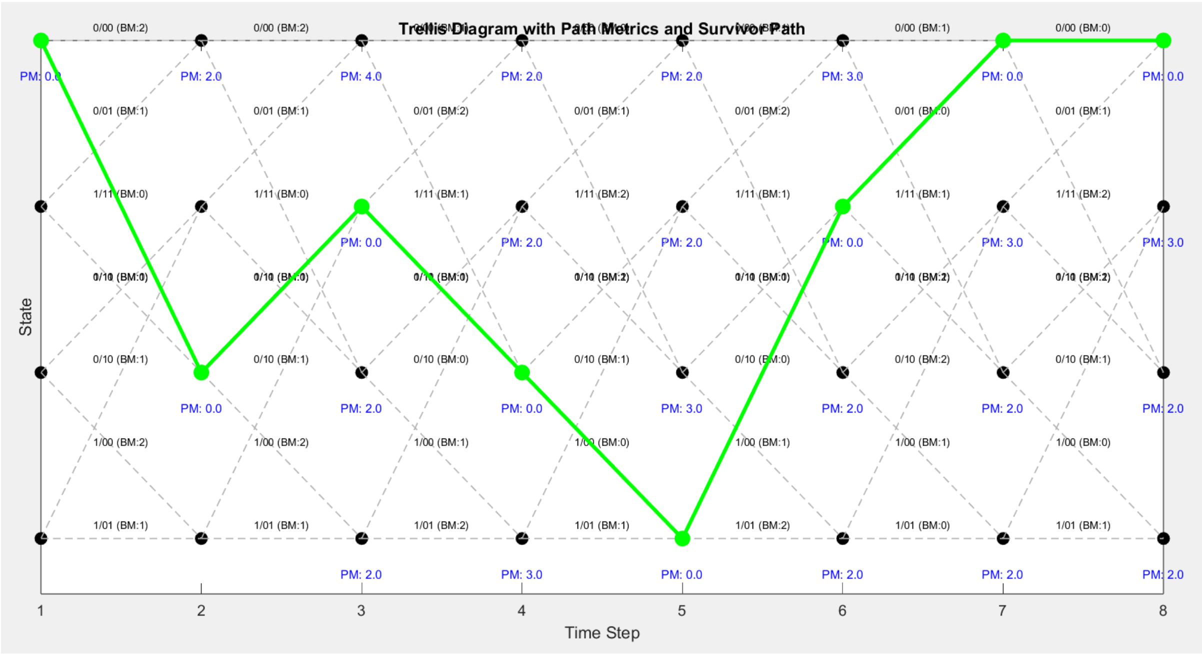
Traceback Process:

Time Step	Current State	Previous State	Input Bit
7	0	0	0
6	0	1	0
5	1	3	0
4	3	2	1
3	2	1	1
2	1	2	0
1	2	0	1

Custom decoder output:
1 0 1 1 0 0 0

✓ CUSTOM DECODER VERIFICATION PASSED

Figure


Trellis Diagram with Path Metrics and Survivor Path
State
Time Step
1
2
3
4
5
6
7
8
PM: 0.0
PM: 2.0
PM: 4.0
PM: 2.0
PM: 2.0
PM: 3.0
PM: 0.0
PM: 0.0
PM: 0.0
PM: 2.0
PM: 2.0
PM: 0.0
PM: 3.0
PM: 3.0
PM: 0.0
PM: 2.0
PM: 0.0
PM: 3.0
PM: 2.0
PM: 2.0
PM: 2.0
PM: 2.0
PM: 3.0
PM: 0.0
PM: 2.0
PM: 2.0
PM: 2.0
0/00 (BM:2)
0/00 (BM:2)
0/00 (BM:1)
0/00 (BM:0)
0/01 (BM:1)
0/01 (BM:1)
0/01 (BM:2)
0/01 (BM:1)
0/01 (BM:2)
0/01 (BM:1)
1/11 (BM:0)
1/11 (BM:0)
1/11 (BM:1)
1/11 (BM:2)
1/11 (BM:1)
1/11 (BM:1)
1/11 (BM:2)
0/10 (BM:1)
0/10 (BM:1)
0/10 (BM:0)
0/10 (BM:1)
0/10 (BM:2)
0/10 (BM:1)
1/00 (BM:2)
1/00 (BM:2)
1/00 (BM:1)
1/00 (BM:1)
1/00 (BM:1)
1/00 (BM:0)
1/01 (BM:1)
1/01 (BM:1)
1/01 (BM:2)
1/01 (BM:1)
1/01 (BM:2)
1/01 (BM:0)
1/01 (BM:1)