Fano Algorithm

Author: Michel Barbeau, Carleton University

Draft version: December 28, 2020

Sequential Decoding with the Fano Algorithm

The Fano algorithm is a sequential form of decoding. It is a suboptimal technique that can handle codes with long constraint lengths. Given a sequence of received bits, decoding is actually a search process. The search is performed in a tree with branches representing messages bits. Each tree node is represented by a pair. The first element is a two-bit register, initialized with 00. The second element is a two-bit codeword associated to the register content. A transition from a node to another represents an input bit, 0 or 1. To obtain the successor state, the left bit is shifted right and the input bit is stored in the left position in the two-bit register. The corresponding codeword is calculated. The root has register 00 and no codeword. Leaf nodes TBD

There is threshold T. The path metric is compared to T. When the path metric is greater that T, corretness of the current path is assumed and the search moves on forward. If the path metric is lower that or equal to T, decoding backtracks for a search of a better path. Thershold T is changing during decoding and brought closer to the path metric, making the path metric always increasing during the search.

```
function [result, cycles, output, newgamma] = fano(scores, maxcycles, K, g, len, mettab
    % Fano algorithm for sequential decoding of convolutional codes
    % Assumptions
    %   1. Coding rate (R) is 1/2.
    %   2. Both polynomials are odd, providing complementary pairs of branch symbols.
    % Inputs
    %   scores = symbol scores (length is 162) - represent input data
    %   result = zero if decoed with sucess
    %   maxcycles = maximum number of cycles
    %   K = constraint
    %   g = code
    %   len = length of output sequence (decoded bits)
    %   mettab = metric table
    % Outputs
    %   result = 0 (success) or 1 (failure)
    %   cycles = actual number of cycles
    %   output = output sequence (decoded data)
    %   newgamma = final metric
    %
    %%% Init threshold
    T = 0;
    %
    %%% Init array of node structures,
    %
    % there is one node per bit in the output sequence
    fprintf('Soft symbols: ');
    for i=1:len
%         fprintf('%d %d ', scores(2*i-1),scores(2*i));
        % initialize all branch metrics
        nodes(i).metrics = zeros(1,4);
        nodes(i).metrics(1) = mettab(1,scores(2*i-1)) + mettab(1,scores(2*i)); % 00
        nodes(i).metrics(2) = mettab(1,scores(2*i-1)) + mettab(2,scores(2*i)); % 01
```

```
        nodes(i).metrics(3) = mettab(2,scores(2*i-1)) + mettab(1,scores(2*i)); % 10
        nodes(i).metrics(4) = mettab(2,scores(2*i-1)) + mettab(2,scores(2*i)); % 11
%          fprintf('Node=%d; %2.2f %2.2f %2.2f %2.2f\n', i, nodes(i).metrics(1), nodes(
        % encoding state, MSB is current bit
        nodes(i).reg = zeros(1,K);
        % cummulative metric, up to that node
        nodes(i).gamma = 0;
        % sorted metrics, will be in decreasing order
        nodes(i).sm = zeros(1,2); % values are arbitrary for now
        % index of current metric being checked,
        % corresponding bit is in "nodes(i).reg(1)"
        nodes(i).k = 1; % highest metric is 1st
    end
    fprintf('\n');
    %
    %%% Init root node
    %
    % metric for cordeword representing bit "0"
    nodes(1).sm(1) = nodes(1).metrics(bi2de(covcoder(zeros(1,K), g),'left-msb')+1);
    % ... metric for cordeword representing bit "1"
    nodes(1).sm(2) = nodes(1).metrics(bi2de(covcoder([zeros(1,K-1) 1], g),'left-msb')+1
    if nodes(1).sm(2)>nodes(1).sm(1) % metric for "1" better than for "0"?
        nodes(1).reg = [zeros(1,K-1) 1]; % try "1" 1st
        temp = nodes(1).sm(1); % switch metrics to have highest first
        nodes(1).sm(1) = nodes(1).sm(2);
        nodes(1).sm(2) = temp;
    end
    %
    %%% Main loop
    %
    % actual number of cyles
    cycles = 0;
    % current node index
    j = 1;
    % main loop
    while cycles <= maxcycles
%          if cycles>4
%              break;
%          end;
        cycles = cycles + 1; % count cycles
        % compute cummulative metric with current branch metric
%          fprintf('Node=%d; %2.2f %2.2f %2.2f %2.2f\n', j, nodes(j).metrics(1), nodes(
%          fprintf("nodes(j).k=%d; nodes(j).sm(nodes(j).k)=%d\n", nodes(j).k, nodes(j).
        newgamma = nodes(j).gamma + nodes(j).sm(nodes(j).k);
%          fprintf("--- Cycle=%d; Node=%d; New gamma=%d, T=%d\n",cycles, j, newgamma, T
%           disp(nodes(j).reg); fprintf("\n");
        if newgamma >= T % new gamma greater than or equal to threshold?
            if nodes(j).gamma < (T+delta) % first time at that node?
                % tighten the threshold
                T = T + delta*floor((newgamma-T)/delta);
            end
            nodes(j+1).gamma = newgamma; % set cummulative metric
            j = j + 1; % move forward
            if j==len+1 % reached the last node?
```

```
                break; % done!
            end
            nodes(j).reg = [nodes(j-1).reg(2:K) 0]; % init encoding state
            % metric for cordeword representing bit "0"
            nodes(j).sm(1) = nodes(j).metrics(bi2de(covcoder(nodes(j).reg, g),'left-msb
            % ... metric for cordeword representing bit "1"
            nodes(j).sm(2) = nodes(j).metrics(bi2de(covcoder([nodes(j-1).reg(2:K) 1], g
%             fprintf(">>> nnodes(j).sm(1)=%d; nodes(j).sm(2)=%d\n", nodes(j).sm(1), no
            if nodes(j).sm(2)>nodes(j).sm(1) % metric for "1" better than for "0"?
                nodes(j).reg = [nodes(j-1).reg(2:K) 1]; % try "1" 1st
                temp = nodes(j).sm(1); % switch metrics to have highest first
                nodes(j).sm(1) = nodes(j).sm(2);
                nodes(j).sm(2) = temp;
            end
%             fprintf(">>> nodes(j).k=%d; nodes(j).sm(nodes(j).k)=%d\n", nodes(j).k, no
%             disp(nodes(j).reg); fprintf("\n");
%             fprintf("<<<\n");
        else % cummulative metric below threshold!
            % backtrack
            while j>=1
            % back to root node?
                if j==1
                    % try again with lower threshold!
                    T = T - delta;
                   if nodes(1).k==2  % was on 2nd best metric?
                       nodes(1).k = 1; % switch back to best metrics
                       nodes(1).reg = [zeros(1,K-1) ~nodes(1).reg(K)]; % flip bit
                    end
                    break; % done backtracking!
                end
                % previous node has metric lower than threshold?
                if nodes(j-1).gamma<T
                    % try again with lower threshold!
                    T = T - delta;
                    if nodes(j).k==2 % was on 2nd best metric?
                        nodes(j).k = 1; % switch back to best metric again
                        nodes(j).reg = [nodes(j).reg(1:K-1) ~nodes(j).reg(K)]; % flip b
                    end
                    break; % done backtracking!
                end
                % back up
                j = j - 1;
                if nodes(j).k == 1 % did 1st metric
                    nodes(j).k = 2; % try 2nd metric
                    nodes(j).reg = [nodes(j).reg(1:K-1) ~nodes(j).reg(K)]; % flip bit
                    break; % done backtracking!
                end
            end
        end
    end
    % generate output sequence
    output = [];
    for i=1:len
        output = [output nodes(i).reg(K)];
```

```
    end
    result = 1;
    if cycles <= maxcycles
        result = 0; % successful decoding
    end
end
```